



UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Extensions and Experimental Evaluation of SAT-based solvers for the UAQ problem

by

Giorgia Gazzarata

Thesis submitted for the degree of *Doctor of Philosophy* (31st cycle)

May 2020

Alessandro Armando	Supervisor
Mauro Giacomini	Supervisor
Giorgio Cannata	Head of the PhD program

Thesis Jury:

Giulio Iannello, <i>Università Campus Biomedico, Rome (Italy)</i>	External examiner
Silvio Ranise, <i>Fondazione Bruno Kessler, Trent (Italy)</i>	External examiner
Luca Viganò, <i>King's College, London (England)</i>	External examiner
Mauro Giacomini, <i>University of Genova, Genoa (Italy)</i>	Internal examiner

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

I would like to dedicate this thesis to my beloved family and to all the friends I neglected
during my Ph.D. studies.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Giorgia Gazzarata
April 2020

Acknowledgements

I would like to acknowledge all the members of CSecLab and Talos for their friendship and important support. I also wish to thank all the people that said to me "you can do it" in difficult times.

Abstract

Nowadays, most of the health organizations make use of Health Information Systems (HIS) to support the staff to provide patients with proper care service. In this context, security and privacy are key to establish trust between the actors involved in the healthcare process, including the patient. However, patients' privacy cannot jeopardize their safety: as a consequence, a compromise between the two must eventually be found. Privilege management and access control are necessary elements to provide security and privacy. In this thesis, we first present the main features that make the Role Based Access Control suitable for permissions management and access control in HIS. We then address the User Authorization Query (UAQ) problem for RBAC, namely the problem of determining the optimum set of roles to activate to provide the user with the requested permissions (if the user is authorized) while satisfying a set of Dynamic Mutually Exclusive Roles (DMER) constraints and achieving some optimization objective (least privilege versus availability). As a first contribution, we show how DMER can be used to support the enforcement of SoD. The UAQ problem is known to be NP-hard. Most of the techniques proposed in the literature to solve it have been experimentally evaluated by running them against different benchmark problems. However, the adequacy of the latter is seldom discussed. In this thesis, we propose a methodology for evaluating existing benchmarks or designing new ones: the methodology leverages the asymptotic complexity analysis of the solving procedures provided in other works to foresee the benchmarks complexity given the values of the most significant RBAC dimensions. First, we use our methodology to demonstrate that the state-of-the-art benchmarks are unsatisfactory. We then introduce UAQ-Solve, a tool that works both as generator of benchmarks and as UAQ solver leveraging existing PMAXSAT complete solvers. By using UAQ-Solve, we apply our methodology to generate a novel suite of parametric benchmarks that allows for the systematic assessment of UAQ solvers over a number of relevant dimensions. These include problems for which no polynomial-time algorithm is known as well as problems for which polynomial-time algorithms do exist. We then execute UAQ-Solve over our benchmarks to compare the performance of different complete and incomplete PMAXSAT solvers.

Contents

List of Figures	viii
List of Tables	xii
Listings	xiii
1 Introduction	1
1.1 Thesis Statement and Organization	4
2 Background	9
2.1 Healthcare and Access Control	9
2.2 Role-Based Access Control	10
2.2.1 Running Example	11
2.2.2 Core RBAC	13
2.2.3 Hierarchical RBAC	14
2.2.4 Constrained RBAC	15
2.3 The User Authorization Query Problem	19
3 Related work	23
3.1 The UAQ problem: definition and complexity	23
3.2 Constraints enforcement	26
4 UAQ problem formalization	30
4.1 Core RBAC	30
4.2 Dynamic Mutually-Exclusive Roles Constraints	31
4.3 The User Authorization Query	33
4.4 Reduction to SS-DMER constraints	37

5 Solving the UAQ problem	41
5.1 Search-based techniques	41
5.2 SAT-based techniques	44
6 Benchmarks	53
6.1 A new approach to UAQ benchmark evaluation and generation	54
6.2 Benchmarks from (64)	60
6.3 The new benchmarks	61
7 A MaxSAT-based solver	66
7.1 UAQ-Solve	66
7.1.1 UAQ-Solve as benchmarks generator	66
7.1.2 UAQ-Solve as benchmarks solver	68
8 Experimental results	82
8.1 Benchmarks from (64)	85
8.1.1 Benchmarks evaluation	86
8.1.2 Solvers evaluation	89
8.2 Our benchmarks with $o_p = \min$	93
8.2.1 Benchmarks evaluation	94
8.2.2 Solvers evaluation	100
8.3 Our benchmarks with $o_p = \max$	106
8.3.1 Benchmarks evaluation	107
8.4 Solvers evaluation through our benchmarks	116
8.5 Use of incomplete solvers	129
8.5.1 Incomplete solvers over hard instances with a small timeout	130
8.5.2 Incomplete solvers over hard instances with timeout 600 seconds .	138
8.5.3 Incomplete solvers over easy instances	139
8.5.4 Summary	142
8.6 Summary of experimental results	143
9 Conclusions and future work	145
9.1 Conclusions	145
9.2 Future work	147
Bibliography	150

Appendix A Proof of Theorem 4.4.1	156
Appendix B WCNF encodings	159
Appendix C Encoding time and number of variables and clauses	171
C.1 Benchmarks with $o_p = \min$	172
C.2 Benchmarks with $o_p = \max$	175

List of Figures

2.1	Use case definition: Richard performing different tasks	12
2.2	Core RBAC (46)	13
2.3	Representation of Core RBAC for the use case	13
2.4	Hierarchical RBAC (46)	15
2.5	Static Separation of Duties in RBAC (46)	16
2.6	Dynamic Separation of Duties in RBAC (46)	17
2.7	Representation of Core RBAC for the use case	21
4.1	Representation of the example	35
8.1	Median solving time of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks presented in Table 6.4 with permission-based optimization .	88
8.2	Median solving time of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks presented in Table 6.4 with joint optimization	91
8.3	$\Delta_{TMedian}$ of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks presented in Table 6.4	92
8.4	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $ Plb $ with permission-based optimization . .	94
8.5	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $ R $ with permission-based optimization . . .	95
8.6	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in \widehat{R}_P with permission-based optimization . . .	96
8.7	Median solving time of UAQ-Solve over the benchmarks parametric in $ P_{ub} $, $ C $, \widehat{rs} and \widehat{t} with permission-based optimization	97
8.8	Median solving time and percentage of skipped problems of UAQ-Solve over Plb_bigR , R_bigPlb and $RPhat_bigPlb$ with different optimization configurations	99

8.9	Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks parametric in $ P_{lb} $ with permission-based optimization	100
8.10	Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks parametric in $ R $ with permission-based optimization	102
8.11	Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks parametric in \widehat{R}_P with permission-based optimization	103
8.12	Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in $ P_{ub} $ with permission-based optimization	104
8.13	Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in $ C $ with permission-based optimization	105
8.14	Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in $\widehat{r}s$ with permission-based optimization	106
8.15	Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in \widehat{t} with permission-based optimization	106
8.16	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $ R $ with permission-based optimization . . .	108
8.17	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmark parametric in $ P_{ub} $ with permission-based optimization . . .	109
8.18	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $ C $ with permission-based optimization . . .	110
8.19	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in \widehat{t} with permission-based optimization . . .	111
8.20	Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $\widehat{r}s$ with permission-based optimization . . .	112
8.21	Median solving time of UAQ-Solve over the benchmarks parametric in \widehat{R}_P and $ P_{lb} $ with permission-based optimization	113
8.22	Median solving time and percentage of skipped problems of UAQ-Solve over R_bigCt with different optimization configurations	114

8.23	Median solving time and percentage of skipped problems of UAQ-Solve over <i>Pub</i> with different optimization configurations	114
8.24	Median solving time and percentage of skipped problems of UAQ-Solve over <i>that_bigRPub</i> with different optimization configurations	115
8.25	Median solving time and percentage of skipped problems of UAQ-Solve over <i>C_bigR</i> with different optimization configurations	116
8.26	Median solving time and percentage of skipped problems of UAQ-Solve over <i>rshat_bigCt</i> with different optimization configurations	116
8.27	Performance of different PMaxSAT solvers over <i>Plb_bigR</i> benchmark . . .	122
8.28	Performance of different PMaxSAT solvers over <i>R_bigPlb</i> benchmark . . .	123
8.29	Performance of different PMaxSAT solvers over <i>RPhat_bigPlb</i> benchmark	124
8.30	Performance of different PMaxSAT solvers over <i>Pub</i> benchmark	125
8.31	Performance of different PMaxSAT solvers over <i>C_bigR</i> benchmark	126
8.32	Performance of different PMaxSAT solvers over <i>rshat_bigCt</i> benchmark .	127
8.33	Performance of different PMaxSAT solvers over <i>that_bigRPub</i> benchmark .	128
8.34	Comparison between MaxHS and LMHS-inc over <i>Plb_bigR</i>	135
8.35	Comparison between MaxHS and LMHS-inc over <i>R_bigPlb</i>	136
8.36	Comparison between MaxHS and LMHS-inc over <i>RPhat_bigPlb</i>	137
8.37	Performance of incomplete solvers over hard benchmarks with timeout 600 seconds	140
8.38	Performance of UAQ-Solve leveraging Loandra and Loandra-inc over the easy benchmarks presented in Tables 6.6 and 6.7 with permission-based optimization	141
C.1	Median encoding time, number of variables and number of clauses in the benchmarks parametric in P_{lb} with $o_p = \min$	172
C.2	Median encoding time, number of variables and number of clauses in the benchmarks parametric in $ R $ with $o_p = \min$	173
C.3	Median encoding time, number of variables and number of clauses in the benchmarks parametric in \widehat{R}_P with $o_p = \min$	174
C.4	Median encoding time, number of variables and number of clauses in the benchmarks parametric in $ P_{ub} $, $ C $, $ \widehat{rs} $, and \widehat{t} with $o_p = \min$	175
C.5	Median encoding time, number of variables and number of clauses in the benchmarks parametric in $ R $ with $o_p = \max$	176

C.6	Median encoding time, number of variables and number of clauses in the benchmarks parametric in $ C $ with $o_p = \max$	177
C.7	Median encoding time, number of variables and number of clauses in the benchmarks parametric in \hat{t} with $o_p = \max$	177
C.8	Median encoding time, number of variables and number of clauses in the benchmarks parametric in \hat{rs} with $o_p = \max$	178
C.9	Median encoding time, number of variables and number of clauses in the benchmarks parametric in $ P_{lb} $ with $o_p = \max$	179

List of Tables

4.1	Actions allowed (\checkmark) or forbidden (\times) by the different kinds of DMER constraints with $rs = \{Doctor, Data_Manager\}$ and $t = 2$	33
4.2	Application of Theorem 4.4.1 on the running example introduced in Section 2.2.1	40
5.1	Complexity of search-based techniques	44
5.2	Conditions on extra permissions and roles in case of joint optimization with $pri = p$	51
5.3	Conditions on extra permissions and roles in case of joint optimization with $pri = r$	51
5.4	Conditions on extra roles in case role-based optimization	52
6.1	Methodology to evaluate benchmarks	56
6.2	Methodology to generate benchmarks with $o_p = min$	58
6.3	Methodology to generate benchmarks with $o_p = max$	59
6.4	Parametric Benchmarks from (64)	60
6.5	Comparing algorithm performances	60
6.6	Benchmark specifications for $o_p = min$	61
6.7	Benchmark specifications for $o_p = max$	63
7.1	Solvers placement at MaxSAT Evaluation 2017 and 2018 for Complete Tracks	78
7.2	Solvers placement at MaxSAT Evaluation 2017 and 2018 for Incomplete Tracks	78

Listings

7.1	Specification file	67
7.2	Instance of UAQ problem generated by UAQ-Solve	68
7.3	WCNF encoding of the instance in Listing 7.2	70
7.4	Summary encoding file for the benchmark generated with Listing 7.1	79
7.5	Plot encoding file for the benchmark generated with Listing 7.1	79
7.6	Summary file for the benchmark generated with Listing 7.1	80
7.7	Plot file for the benchmark generated with Listing 7.1	80
7.8	Quantiles file for the benchmark generated with Listing 7.1	80
7.9	Solving times of different complete solvers over the benchmark generated with Listing 7.1	81
7.10	Solution costs over the benchmark generated with Listing 7.1	81
B.1	WCNF encoding for $o_p = \min$, $o_r = \text{any}$, and $pri = p$	159
B.2	WCNF encoding for $o_p = \max$, $o_r = \text{any}$, and $pri = p$	160
B.3	WCNF encoding for $o_p = \min$, $o_r = \min$, and $pri = p$	163
B.4	WCNF encoding for $o_p = \min$, $o_r = \max$, and $pri = p$	164
B.5	WCNF encoding for $o_p = \max$, $o_r = \min$, and $pri = p$	166
B.6	WCNF encoding for $o_p = \max$, $o_r = \max$, and $pri = p$	167
B.7	WCNF encoding for $o_p = \min$, $o_r = \min$, and $pri = r$	169

Chapter 1

Introduction

Nowadays, most of the health organizations make use of Health Information Systems (HIS) to support the staff to provide patients with proper care service. In this context, security and privacy are key to establish trust between the actors involved in the healthcare process, including the patient. Privilege management and access control are necessary elements to provide security and privacy. The purpose of privilege management is to define the permissions of a user, following policies and possibly taking into account environmental and contextual constraints. Based on that, the requested access is permitted or denied. Permissions must be managed in a way that only the medical staff involved in a patient care can access the patient's clinical information, according to the 'need to know' principle. However, data protection measures should not prevent the appropriate use of care data: because the patient life is more important than preserving her privacy, it is crucial that healthcare data are always available when needed. As a consequence, usability and data availability are fundamental features for the system to be accepted and adopted. In particular, a trade off between patients privacy and safety should always be reached.

The Role-Based Access Control (RBAC) is one of the most prominent access control models. It is widely accepted as a best practice for privilege management within a single system or application. Its popularity is demonstrated by direct or indirect use in different commercial products and organizations. The success of RBAC is mainly due to the simple administration and the ability to implement advanced security principles like least privilege and separation of duties.

The use of RBAC implies the activation and deactivation of roles within a session. In systems offering *permission-level* user-system interaction, the system automatically determines the roles that need to be activated in order to grant the requested permissions. This kind of

interaction opposes to the *role-level* interaction, where the user explicitly tells the system which roles must be activated.

In the healthcare context, it is fundamental to provide permission-level user-system interaction for reasons of usability and optimality of roles activation. In this dissertation, we address the User Authorization Query (UAQ) problem for RBAC, which is key in systems offering permission-level user-system interaction. The UAQ problem consists in determining the optimum set of roles to activate to provide the user with the requested permissions (if the user is authorized) while satisfying a set of Dynamic Mutually Exclusive Roles (DMER) constraints and achieving some optimization objective (least privilege versus availability). DMER constraints are motivated by the least privilege, while they are not adequate to enforce separation of duties. To overcome this problem, an extension of DMER has been put forward (13). As a first contribution, we demonstrate that the extended DMER constraints can be reduced to equivalent traditional DMER constraints.

The UAQ problem is known to be NP-hard. Many works in the literature provide techniques to solve the UAQ problem. However, none of them provides a valuable and satisfactory experimental evaluation. Most of the UAQ solvers have been experimentally evaluated by running them against different benchmark problems. The latter are commonly parametric in one of the aspects of the UAQ problem that may contribute to its complexity. All the other aspects are either set to a predefined, constant value or are randomly chosen in a given interval or according to some other criterion. Unfortunately, the state-of-affairs is unsatisfactory. For example, it happens that the experimental results exhibit polynomial solving time when a polynomial-time technique is not known. This means that the benchmarks used to test the solvers are not representative of the UAQ problem complexity. As a consequence, the actual efficiency of the solutions proposed is not clear.

It is then obvious that we need a suitable set of benchmarks for concretely assessing UAQ solvers. However, to our knowledge, a systematic approach to benchmarking still does not exist. In this thesis, we propose a methodology for evaluating existing benchmarks or designing new ones: the methodology leverages the asymptotic complexity analysis of the solving algorithms provided in (64) to foresee the complexity of the benchmark given the values of the most significant RBAC dimensions. The aforementioned algorithms play a key role in the proposed methodology, since they provide upper bounds on the asymptotic growth rate for UAQ solvers which, to the best of our knowledge, are the best upper bounds currently available in the literature. Yet, it must be noted that the proposed methodology will yield different, improved results as soon as new complexity results will become available. To the

best of our knowledge, this is the first approach to systematically assess the effectiveness of UAQ solvers.

First, we use our methodology to demonstrate that state-of-the-art benchmarks are unsatisfactory. We then introduce UAQ-Solve, a tool that works both as a generator of benchmarks and as a UAQ solver. UAQ-Solve reduces the UAQ problems to PMaxSAT problems and then leverages existing PMaxSAT solvers. By using UAQ-Solve, we apply our methodology to generate a novel, complete, and satisfactory suite of parametric benchmarks that allows for the systematic assessment of UAQ solvers over a number of relevant dimensions. Each benchmark is parametric in one aspect characterizing the UAQ problem complexity, while the other dimensions are fixed. The benchmarks proposed and used in this thesis could be improved in the future in consequence to the improvement of the methodology due to new complexity results.

Afterward, we execute UAQ-Solve over our benchmarks leveraging different PMaxSAT solvers to compare their performance. The solvers exhibit the same behavior over the same benchmark, and there is not a general best solver. Along the same dimension, the solver that best performs for small values of the parameter may be among the worst solvers for big values of the same parameter. Over different benchmarks, the solver that outperforms the others over a benchmark may be particularly inefficient over another benchmark. UAQ-Solve over our benchmarks can then be of help to carefully choose the proper solver to use to tackle the UAQ problem.

Because solving certain UAQ problems can take a large amount of time and patients privacy cannot jeopardize their safety, a trade-off between the two must be put forward. As a final contribution we then make some investigations on incomplete solvers. The latter are solvers that provide the best solution found if an optimum solution is not known by a pre-set timeout. The experiments reveal that the quality of the solutions provided by an incomplete solver after 1 second over hard problems seems to be good. In addition, it seems that the use of an incomplete solver over easy problems is not detrimental to efficiency. Overall, the incomplete solvers offer a good compromise between the quality of the solutions and usability. Consequently, they are suitable for all the contexts in which usability can not be set aside in favor of security, like healthcare, in our case. To the best of our knowledge, we are the first ones who propose the use of incomplete solvers over UAQ problems to tackle usability.

1.1 Thesis Statement and Organization

The UAQ problem is known to be NP-hard (28; 35). Many works in the literature provide techniques to solve it. However, none of them provides a valuable and satisfactory experimental evaluation. Most of the techniques have been evaluated by running them against different benchmark problems. These benchmarks are usually parametric in some relevant dimension of the problem (e.g., number of roles, number of DMER constraints, number of requested permissions) and aim at evaluating the scalability of the proposed techniques along them. Unfortunately, the state-of-affairs is unsatisfactory. The available benchmarks do not cover (and thus do not test the solvers against) all the relevant aspects of the problem. For instance, the problems used in (64) do not consider the case where the number of permissions to be activated is maximized (i.e., $\text{obj}=\text{max}$): they therefore do not allow the evaluation of the respective UAQ instances. Furthermore, solvers are often evaluated against problems proposed by the same authors and this does not permit to assess the relative merits of the proposed techniques. In addition, it happens that the experimental results exhibit polynomial solving time when a polynomial-time technique is not known, see for example (90). This could actually mean that:

- the benchmark represents a tractable subclass of UAQ problems;
- the range of the parameter under test is not wide enough to highlight the expected behavior; or
- the algorithm implemented by the solver is more efficient than the best algorithms known over the class of problems under test;
- $P = NP$. It is possible, but highly unlikely.

As a consequence, the actual efficiency of the solutions proposed is not clear. Even more so, for the same reason, it is not easy to compare the performance of the different techniques proposed to tackle the UAQ problem.

This thesis provides the following contributions:

- We demonstrate that any UAQ problem leveraging the extended DMER constraints can be reduced to an “equivalent” UAQ problem containing only the traditional DMER constraints provided that a succinct representation of the current and past role activations is available (Section 4.4). These results enable the use of existing UAQ solvers to support the enforcement of SoD requirements;

- By leveraging the asymptotic complexity analysis of the solving procedures provided in (64), we propose a methodology for designing parametric benchmarks for the UAQ problem (Section 6.1). The methodology leads to benchmarks capable to (i) stress-test solvers along the dimensions of the problem for which no polynomial-time technique is known but also (ii) to check their effectiveness, by determining whether they efficiently solve problems that are known to be solvable in polynomial time. To our knowledge, we are the first who propose a scientific and rigorous methodology to evaluate and design benchmarks. As we will discuss in Section 6.1, to make the methodology practically applied, we make two important assumptions:
 1. Any benchmark is parametric in a specific dimension. In principle, the latter is *infinite*. We will observe the behavior of different solvers in a *finite* interval of the parameter under test, assuming that the solvers have the same behavior outside the defined interval;
 2. The methodology would require to run all the possible UAQ problem solvers over the benchmarks. Since in principle the solvers could be *infinite*, we consider only a *finite* subset of them. We then assume that the other solvers behave similarly to the ones considered in our evaluation.
- We present UAQ-Solve, which is both a benchmarks generator and a solver (Section 7.1). In combination with our methodology, the tool can generate hard benchmarks of UAQ problems. The latter can constitute important input for the MaxSAT Evaluation: in fact, in addition of being hard, the benchmarks are relevant from the application point of view. We already submitted some benchmarks to the MaxSAT Evaluation 2018 (10), where our benchmarks were used to stress-test solvers participating to both the complete and incomplete tracks ¹;
- By using our methodology, we introduce a novel suite of parametric benchmarks that allows for the systematic assessment of UAQ solvers over a number of relevant dimensions (Section 6.3). These include problems for which no polynomial-time algorithm is known as well as problems for which polynomial-time algorithms do exist.

¹ Results of Unweighted Complete Track Per Benchmark Family (you can find the results over our benchmarks searching for "uaq"): <https://maxsat-evaluations.github.io/2018/results/complete/unweighted/table-extended.html>. Results of Unweighted Incomplete Track under 60 and 300 seconds Timeout (you can find the results over our benchmarks searching for "uaq"): <https://maxsat-evaluations.github.io/2018/results/incomplete/unweighted-60s/table-all.html> and <https://maxsat-evaluations.github.io/2018/results/incomplete/unweighted-300s/table-all.html>.

Guided by the asymptotic complexity analysis results given in (64), we indicate, for each benchmark, its purpose and the expected behavior of solvers. To our knowledge, we are the first who provides a set of benchmarks designed through a methodology like the one we present. This contribution is particularly important. In fact, the lack of adequate benchmarks makes it difficult to assess the efficiency of the different techniques used to tackle the UAQ problem. Even more so, it makes it difficult to compare their performance. By providing a complete and adequate suite of benchmarks, we are de facto offering the possibility to both

- assess the performance of a single solver, and
- compare the performance of the different solutions.

(69) provided a similar contribution for the famous Symmetric and Asymmetric Traveling Salesperson Problems (TSP). This work helped to

- reduce the publication of relatively weak TSP heuristics, and
- motivate development of new TSP approaches;
- We use the generated benchmark to evaluate the performance of different solvers, among which PMaxSAT solvers (Sections 8.2.2 and 8.4);
- We show that the incomplete solvers over hard benchmarks can provide sub-optimal solutions that are a good compromise between the optimization objective and usability (Section 8.5). In fact, contrary to the complete solvers, if the incomplete solvers cannot find the optimum solution within a preset timeout, they simply return the best solution found. Besides, we also show that the incomplete solvers can also perform similarly to the complete solvers over easy benchmarks. To our knowledge, we are the first ones who propose the use of incomplete solvers to tackle the UAQ problem.

We have used the new suite of benchmark problems as well as the benchmarks introduced in (64) to experimentally evaluate three solvers:

- 2D-Opt-Search (64): a search-based solver;
- 2D-Opt-CNF (64): combines the reduction of the UAQ Decision Problem to SAT, a state-of-the-art SAT solver and a binary search;
- UAQ-Solve: a SAT-based solver that implements a reduction of the UAQ problem to PMaxSAT and uses any state-of-the-art PMaxSAT solver to tackle the problem. It is an extension to (13).

UAQ-Solve outperforms both 2D-Opt-Search and 2D-Opt-CNF in the vast majority of the benchmark considered. This is hardly surprising since PMaxSAT solver implements sophisticated search algorithms specifically tailored to tackle optimization problems; in contrast, 2D-Opt-CNF tackles the optimization problem through a fairly naive binary search strategy and 2D-Opt-Search simply enumerates the solutions in order to find an optimum. Yet, prior to our experimental analysis, there was little evidence (if any) to support this conclusion.

Besides, we executed UAQ-Solve over the proposed benchmarks leveraging different PMaxSAT solver to compare their performances.

The remainder of this dissertation is organized as follows:

Chapter 2 introduces background information;

Chapter 3 presents the related work;

Chapter 4 shows the formalization of RBAC and the UAQ problem together with the reduction of the extended DMER introduced in (13) to traditional DMER constraints;

Chapter 5 presents the different techniques to solve the UAQ problem;

Chapter 6 explains the methodology we propose to evaluate/generate benchmarks and introduces state-of-the-art benchmarks and our benchmarks. This chapter presents and extends the content of our paper² (12);

Chapter 7 introduces our tool, UAQ-Solve. The latter is used to generate the benchmarks designed in Chapter 6 and to run the experiments presented in Chapter 8. The tool is also discussed in (11)³.

Chapter 8 shows the experimental results. In particular:

Sections 8.1 demonstrates that state-of-the-art benchmarks are incomplete and unsatisfactory. In addition, it shows that UAQ-Solve outperforms 2D-Opt-Search and 2D-Opt-CNF over them;

Sections 8.2 and 8.3 contain the evaluation of our benchmarks. Besides, it shows that UAQ-Solve outperforms 2D-Opt-Search and 2D-Opt-CNF over them;

Section 8.4 compares the performance of different PMaxSAT solvers over our benchmarks;

²Accepted at SACMAT 2020.

³Accepted at SACMAT 2020. Here we decided to rename the tool "AQUA".

Section 8.5 shows the results of the execution of incomplete solvers over our benchmarks;

Chapter 9 exposes the conclusions and future works.

Chapter 2

Background

2.1 Healthcare and Access Control

Nowadays, most of the health organizations make use of Health Information Systems (HIS) to support the staff to provide patients with proper care service. In this context, security and privacy are key to establish trust between the actors involved in the healthcare process, including the patient (16). Security aims at guaranteeing information availability, confidentiality, integrity, authenticity and accountability. Privacy is a human right for self-determination. It requires compliance with legal requirements, ethical principles, personal preferences and expectations regarding the treatment of personal data (22). Privilege management and access control are key elements to provide security and privacy. In a healthcare business process, a user wants to access clinic information. Privilege management and access control deal with the relationship between a subject (actor, user) and an object (resource, information), thereby defining the privileges (permissions) the entity obtains regarding the object. Permissions must be managed in a way that only the medical staff involved in a patient's care can access the patient's clinic information according to the 'need to know' principle. Access implies different operations on that object such as create, read, write, update, delete, execute. The purpose of privilege management is to define the permissions a user has, following policies and possibly taking into account environmental and contextual constraints. The policies define which behaviors are permitted or forbidden to the subjects, as well as constrain actions. For example, a doctor can retrieve patients' health data, but she can not disclose them. Based on policies and other eventual constraints, the requested access is permitted or denied. However, data protection measures should not prevent the appropriate use of care data: security and privacy requirements should not affect the usability of the entire system and healthcare data availability. Because the patient's life is more important than preserving her privacy, usability

is in fact a fundamental feature for the system to be accepted and adopted. In this thesis, we propose the Role-Based Access Control as access control model, for several reasons that will be introduced in Section 2.2.

2.2 Role-Based Access Control

Role-Based Access Control (RBAC) is an access control model introduced in 1992 in (39). Although developed by the National Institute of Standard and Technology (NIST), the International Committee for Information Technology Standards (INCITS) adopted, copyrighted, and distributed RBAC as INCITS 359-2004 (46). RBAC is widely accepted as a best practice to manage user privileges within a single system or application. Its popularity is demonstrated by the direct or indirect use in different products and organizations. To name few: Apache Fortress Rest and Apache Fortress Web¹, Kubernetes², Microsoft Office 365 and Microsoft DefenderAdvanced Threat Protection³, Microsoft Windows Server (from 2003)⁴, Microsoft Azure⁵ and other Microsoft products, RedHat OpenShift⁶, Apache Ignite⁷, and Oracle Solaris OS⁸. Besides, (62) analyzed the economic value of RBAC for enterprises, and estimated benefits per employee from reduced employee downtime, more efficient provisioning, and more efficient access control policy administration.

The central concept of RBAC is the separation between users and the permissions to access the resources. This is pursued through role relations: a role is “a collection of permissions to use resources appropriate to a person’s job function” or competencies to do specific tasks in an organization (88). Instead of specifying all the accesses each user is allowed to execute, permissions are assigned to roles; then, users are assigned to roles based on their authorities, responsibilities, and qualifications. Consequently, access decisions are based on the roles that individual users have as part of an organization. Often the associations between users and permissions are dynamic. Instead, roles are more stable, because an organization’s activities tend to remain relatively constant or change slowly over time (71). For this reason, RBAC greatly simplifies the management of users’ permissions. In fact, the

¹<https://directory.apache.org/fortress/overview.html>

²<https://kubernetes.io/docs/reference/access-authn-authz/rbac/>

³<https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/rbac>

⁴<https://docs.microsoft.com/en-us/windows-server/networking/technologies/ipam/role-based-access-control>

⁵<https://docs.microsoft.com/en-us/azure/role-based-access-control/overview>

⁶<https://docs.openshift.com/dedicated/4/authentication/using-rbac.html>

⁷<https://apacheignite.readme.io/docs/rbac-authorization>

⁸https://docs.oracle.com/cd/E63708_01/doc.801/e63712/mssg_rbac.htm#MSVSG265

administrative task consists of granting, reassigning, and revoking users' membership to the set of roles within the system:

- When a new person enters the organization, the administrator simply grants membership to an existing role;
- When a person's function changes within the organization, the administrator deletes his/her membership to existing roles and creates new ones;
- When a person leaves the organization, the administrator deletes all his/her memberships to all roles.

Also, roles can be updated without updating the privileges for every user on an individual basis. In addition, roles can be associated to new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed (39) (71).

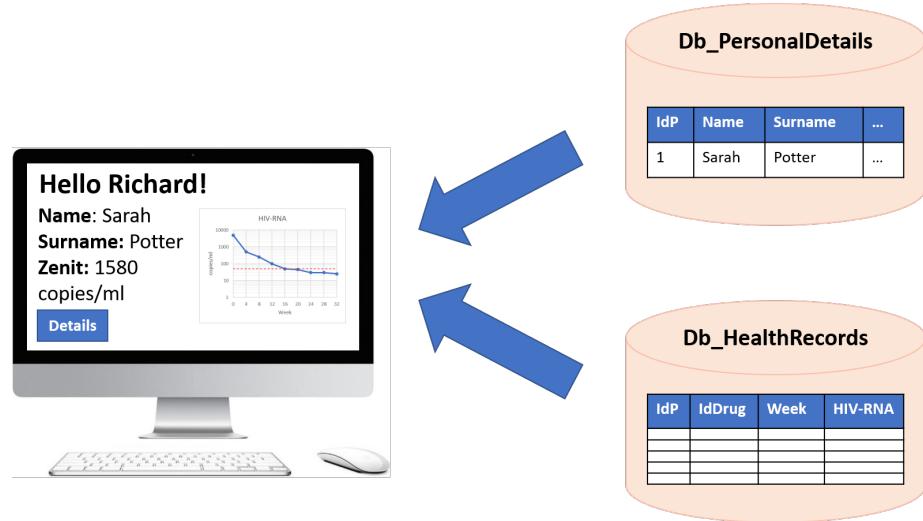
Another RBAC strength is the possibility to enforce the least privilege principle, which is important for meeting integrity objectives (84). The RBAC reference model is defined in terms of different model components: Core RBAC, Hierarchical RBAC, and Constrained RBAC (Static Separation of Duty and Dynamic Separation of Duty RBAC) (46). In the following subsections, we review the aforementioned RBAC models with the help of the simple use case described below.

2.2.1 Running Example

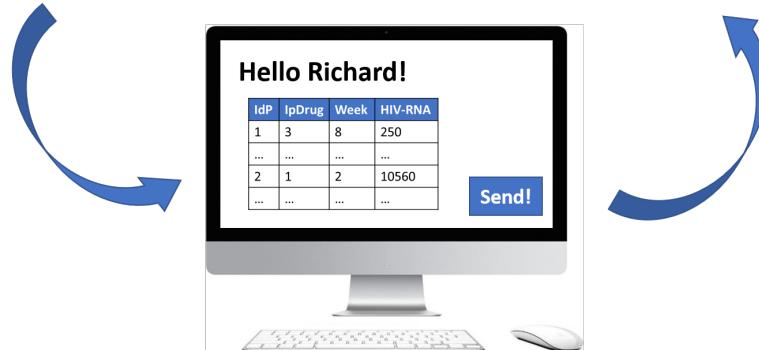
Richard is a doctor working in the Department of Infectious Diseases of a medical facility in which Sarah, HIV-infected, is hospitalized. To provide Sarah with health care service, Richard needs to access her health records through the HIS. In order to minimize the risk of personal information disclosure, the HIS makes use of two databases hosted in two distinct servers: one database stores the patients' personal details and the other one stores the patients' health records (accessible through the patients' identifier). Consequently, as shown in Figure 2.1a, Richard needs to retrieve Sarah's identifier from the first database; then, he can retrieve Sarah's health records from the second one thorough Sarah's identifier.

Besides, as shown in Figure 2.1b, Richard is a Data Manager responsible for providing anonymized information to a pharmaceutical company to make some statistics on the patient's reactions to new drugs. This task is critical: in fact, if not managed properly, it could lead to personal data disclosure. This would have serious consequences: it would be bad publicity for the hospital, and, above all, Sarah could be denigrated because of her illness.

Jane is the pharmacist who provides medicines in the department. Claire is a nurse and needs to access Richard's prescription to administer drugs to Sarah. Finally, Matthias is a doctor and data manager like Richard. However, Matthias is also the head physician, who, among the other things, manages the staff schedule and checks that the staff follows the guidelines during the healthcare service.



(a) Richard acting as doctor



(b) Richard acting as data manager

Figure 2.1 Use case definition: Richard performing different tasks

2.2.2 Core RBAC

Figure 2.2 shows Core RBAC, which is the base of all RBAC systems. Core RBAC consists of the minimum collection of data elements to achieve a complete RBAC system: users, roles, permission, operations, and objects. Permissions (PRMS) are approval to perform operations (OPS) on protected objects (OBS). Every permission can be associated to one or more roles through Permission Assignment (PA) relations. Similarly, every role can be associated to one or more users through User Assignment (UA) relations. Users who are members of a certain role acquire all its permissions. A single user can have one or more roles and a role can be associated to one or more users (88). In addition, the core RBAC model includes the sessions set; the latter maps each user to the activated subset of roles assigned to her in the current session. Each session is associated with a single user and each user is associated with one or more sessions. The permissions actually available to the user are the ones assigned to the roles activated in the user's sessions (46).

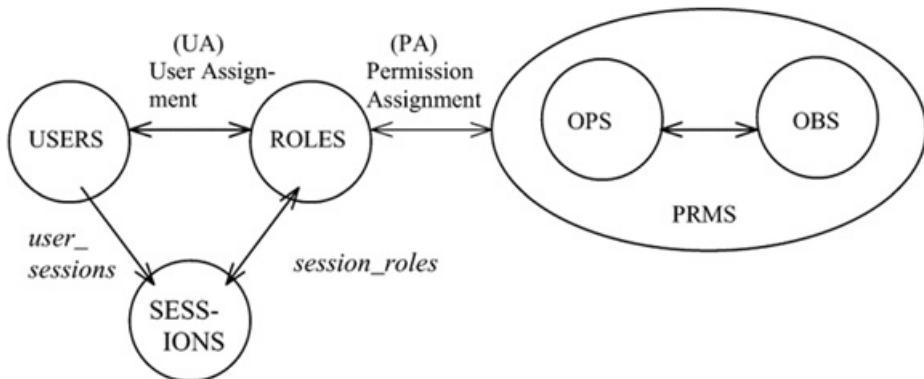


Figure 2.2 Core RBAC (46)

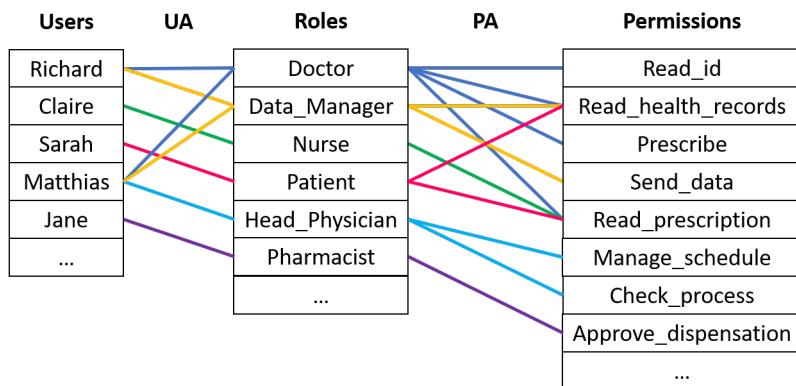


Figure 2.3 Representation of Core RBAC for the use case

Figure 2.3 shows the representation of the Core RBAC model for our use case. Through the activation of the role *Doctor*, Richard can get the identification of Sarah and her personal details (*Read_id*), read her health records (*Read_health_records*), and prescribe drugs to Sarah (*Prescribe*). Instead, through the activation of the role *Data_Manager*, Richard can retrieve patients' health record (*Read_health_records*) and send them as anonymized data to the pharmaceutical company (*Send_data*). Claire, through the activation of the role *Nurse*, can read Richard's prescription to administer drugs to Sarah (*Read_prescription*) when she is hospitalized. Sarah, through the activation of role *Patient* can read her own health records (*Read_health_records*) and read Richard's prescription (*Read_prescription*). Finally, Matthias can perform the same operations as Richard, but, in quality of *Head_Physician*, he can also manage the staff schedule (*Manage_schedule*) and check if the staff follows the guidelines during the Sarah's care process (*Check_process*). Finally, Jane, as *Pharmacist*, can approve the provisioning of medicines (*Provide_medicines*).

2.2.3 Hierarchical RBAC

Within many organizations, there are a number of general permissions that should be granted to a large number of users; furthermore, only senior employees should perform more critical operations. To reflect an organization's lines of authority and responsibility, RBAC models include the concept of role hierarchies (88). Formally, a hierarchy is a partial order defining an inheritance relation between roles: for example, if role r_1 inherits role r_2 , then all privileges of r_2 are also privileges of r_1 (46). In our example in Figure 2.3, both the roles *Doctor* and *Nurse* have the permission *Read_prescription* assigned. In addition, *Doctor* has assigned other permissions that are not assigned to *Nurse*. Consequently, it is possible to define a hierarchical relation between *Doctor* and *Nurse* where *Doctor* is the most senior role. Figure 2.4 shows Hierarchical RBAC, which consists of the Core RBAC with the addition of requirements for supporting role hierarchies. Since the hierarchical relations are not relevant for our scopes, we do not introduce more details.

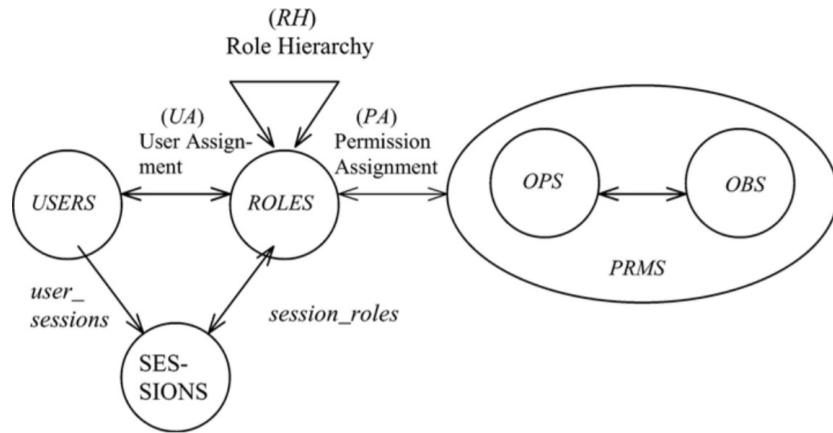


Figure 2.4 Hierarchical RBAC (46)

2.2.4 Constrained RBAC

The Separation of Duties (SoD) is a security principle used to avoid conflicts of interest or errors. Its purpose is to ensure that frauds and major errors within an organization are caused only as a result of deliberate collusion among multiple individuals. SoD has long been recognized for its wide application in business, industry, and government (46).

With respect to Core RBAC, the Constrained RBAC provides for SoD relations to enforce conflict of interest policies. The latter prevent users from exceeding a reasonable level of authority for their positions. There are two major types of SoD relations, namely Static SoD (SSD) and Dynamic SoD (DSD) relations, which are described in the following.

SSD and Static Mutually Exclusive Roles constraints

A conflict of interest or errors may occur when a user gains authorization for permissions associated with conflicting roles. As shown in Figure 2.5, Static SoD relations can place restrictions on sets of roles and, in particular, on their ability to form UA relations (46). In case of role hierarchy, special care must be applied to ensure that user inheritance does not undermine SoD policies.

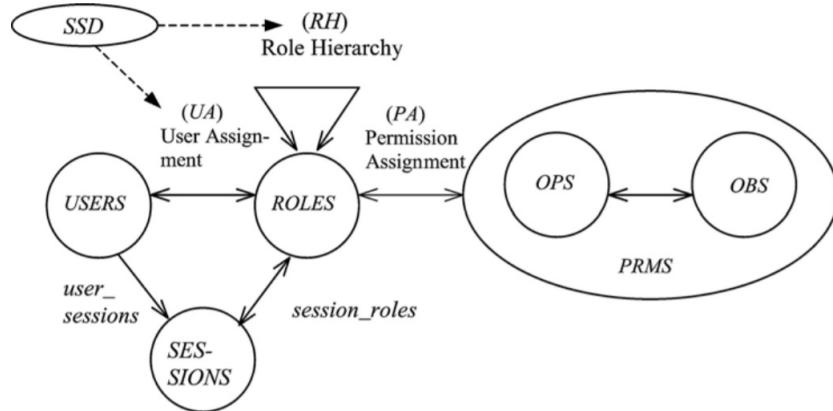


Figure 2.5 Static Separation of Duties in RBAC (46)

In RBAC, the Static SoD policies can be enforced through the Static Mutually Exclusive Roles (SMER) constraints. A SMER constraint is defined by mutually disjoint user assignments with respect to a set of m roles $\{r_1, r_2, \dots, r_m\}$ and by an associated cardinality t . The constraint

$$\text{SMER}(\{r_1, r_2, \dots, r_m\}, t)$$

means that no more than $t - 1$ roles in $\{r_1, r_2, \dots, r_m\}$ can be assigned to a user.

In our example, Claire, as a *Nurse*, can administer medicines; it would be unsafe if she could also be assigned the role *Pharmacist*, which would enable her to approve the dispensation of medicines. In fact, Claire could get hold of dangerous medicine and administer it to a patient. In this case, the following SMER can be used:

$$\text{SMER}(\{\text{Nurse}, \text{Pharmacist}\}, 2).$$

From a policy perspective, SMER constraints are a powerful mean to enforce conflict of interest rules over sets of roles. However, their definition can often be excessively restrictive. In our example, the simultaneous activation of the roles *Doctor* and *Data_Manager* is dangerous: in fact, Richard could accidentally (or not) send the patients personal details contained in the health record to the pharmaceutical company, causing personal information disclosure. On the other hand, the constraint

$$\text{SMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)$$

would not be adequate: in fact, a person may be both a doctor and a data manager for the same hospital.

It is important to recall the fundamental difference between Separation of Duties policies and Static MER constraints. In fact, Static SoD policies are *objectives* that need to be achieved to prevent frauds and major errors: specifically, a SSoD policy specifies the minimum number of users allowed to possess together all the permissions needed for a sensitive task. It is then clear that Static SoD policies do exist independently of the access control model used to manage the permissions. On the contrary, SMER constraints are specific to RBAC: in particular, SMER constraints are the *mechanism* that RBAC provides to enforce SSoD policies. With the SMER constraints, the SSoD policies are achieved by limiting the role memberships a single user is allowed to have. Obviously, to ensure SMER constraints effectiveness, it is necessary to assign permissions to roles properly. In fact, for example, a SMER constraint can not enforce a given SSoD policy if all the permissions needed to perform a sensitive task are assigned to a single role. As shown in (51), directly enforcing the SSoD policies is intractable (coNP-complete), while enforcing SMER constraints is efficient; however, verifying whether a given set of SMER constraints enforces the SSoD policies is still intractable (coNP-complete).

DSD and Dynamic Mutually Exclusive Roles constraints

Like SSD relations, Dynamic SoD relations enable to limit the permissions that are available to a user. However, the context in which these limitations are imposed is different: in fact, while SSD relations restrict the ability to form UA relations, DSD relations constrain the roles that are activated in a user's session, as shown in Figure 2.6.

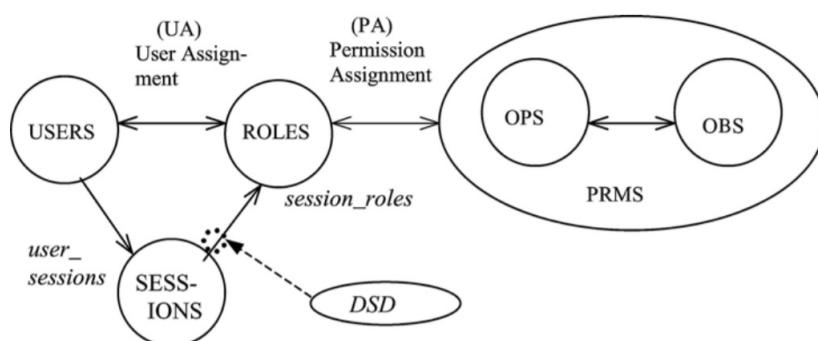


Figure 2.6 Dynamic Separation of Duties in RBAC (46)

As a consequence, DSD relations enable a user to be authorized for two or more roles that do not create a conflict of interest or possible errors when acted independently, but forbid a simultaneous activation.

DSD relations can support the least privilege principle, ensuring that user's permissions do not persist beyond the time that they are required for performance of duty. This aspect of least privilege is often known as timely revocation of trust and it would be a rather complex issue without the definition of DSD relations (46).

Within RBAC, DSD policies can be enforced through Dynamic Mutually Exclusive Roles (DMER) constraints. A DMER constraint requires that no user can activate t or more roles in a set of m roles $\{r_1, r_2, \dots, r_m\}$ *in any session*.

In our example, the constraint

$$\text{DMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)$$

would enable Richard to carry out the employment related to one role between *Doctor* and *Data_Manager* per time, but would deny a simultaneous activation of both the roles in the same session.

Although more flexible, DMER constraints are not suitable for enforcing SoD policies, as pointed out in (51). In fact, Richard could activate *Doctor* and *Data_Manager* sequentially in any session; namely he could:

1. activate *Doctor*;
2. deactivate *Doctor*;
3. activate *Data_Manager*.

Otherwise, he could activate *Doctor* and *Data_Manager* simultaneously in two different sessions. This is a serious limitation as SoD policies play a key role in preventing frauds and misuse of the system resources. To overcome this important issue, (13) introduced four different kinds of MER constraints, which are briefly described in the following paragraphs.

Single-Session DMER constraints ensure that no user can simultaneously activate t or more roles in a set of m roles $\{r_1, r_2, \dots, r_m\}$ *in any session*. The constraint

$$\text{SS-DMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)$$

is the common DMER constraint, already described above.

Multi-Session DMER constraints ensure that no user can simultaneously activate t or more roles in a set of m roles $\{r_1, r_2, \dots, r_m\}$ in all the sessions owned by the user. The constraint

$$\text{MS-DMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)$$

would deny the simultaneous activation of *Doctor* and *Data_Manager* in one or more sessions, but Richard still could activate them sequentially.

Single-Session History-based DMER constraints ensure that no user can activate t or more roles in a set of m roles $\{r_1, r_2, \dots, r_m\}$ in any session over time. The constraint

$$\text{SS-HMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)$$

would deny the sequential activation of *Doctor* and *Data_Manager*, however Richard still could activate them in two different sessions.

Multi-Session History-based DMER constraints ensure that no user can activate t or more roles in a set of m roles $\{r_1, r_2, \dots, r_m\}$ in all the sessions owned by the user over time. The constraint

$$\text{MS-HMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)$$

would forbid both the sequential and the simultaneous activation of *Doctor* and *Data_Manager* in one or more sessions.

These extensions would enable the DMER constraints to enforce SoD policies, while preserving their flexibility.

2.3 The User Authorization Query Problem

The User Authorization Query (UAQ) Problem for RBAC amounts to determining an optimum set of roles to activate in a given session in order to obtain some permissions while satisfying a collection of authorization constraints, most notably Dynamic Mutually-Exclusive Roles (DMER) constraints. The UAQ problem is key in systems offering *permission-level* user-system interaction as opposed to *role-level* interaction. In the former, the system automatically determines the roles to activate to enable the requested permissions. In contrast, in

the latter, the user explicitly determines and tells the system which roles must be activated. Permission-level user-system interaction is fundamental in applications in which usability is an important specification.

In the authorization query, the requested permissions come in two sets: a lower bound P_{lb} and an upper bound P_{ub} such that $P_{lb} \subseteq P_{ub} \subseteq P$, where P is the complete set of permissions. The permissions in P_{lb} are those that *must* be granted, whereas those in $P_{ub} \setminus P_{lb}$ are additional permissions that *may* be granted. It is then possible to either minimize or maximize the number of additional permissions to be granted depending on which security objective (safety or availability, respectively) needs to be prioritized. If safety (availability) is more important, then the number of permissions from $P_{ub} \setminus P_{lb}$ needs to be minimized (maximized, respectively). Notice that a certain degree of safety is achieved even if availability is preferred over safety, since no permission in $P \setminus P_{ub}$ can be granted. In our example, suppose that Matthias needs to plan the staff schedule for the next month: to perform this task, he needs the permission *Manage_schedule*, thus $P_{lb} = \{\text{Manage_schedule}\}$. On the contrary, he does not need the permission *Send_data*, whose activation could be risky: consequently, we could exclude it from P_{ub} , that could be $P_{ub} = \{\text{Read_id}, \text{Read_health_records}, \text{Prescribe}, \text{Read_prescription}, \text{Manage_schedule}, \text{Check_process}\}$. In this situation:

- *Manage_schedule* must be granted, because it is the requested permission;
- *Send_data* must be denied, because it is in $P \setminus P_{ub}$;
- *Read_id*, *Read_health_records*, *Prescribe*, *Read_prescription*, and *Check_process* are in $P_{ub} \setminus P_{lb}$ and consequently:
 - if the objective is safety, we would like not to activate them;
 - if the objective is availability, we would like to activate them.

Following the aforementioned conditions, in case the objective is safety, the optimum solution is the activation of the only *Head_Physician*, which provides Matthias with the permissions *Manage_schedule* (namely, the requested permission) and *Check_Process*. On the contrary, if the objective is availability, the optimum solution is the activation of the roles *Head_Physician* and *Doctor*. The latter activates the permissions *Read_id*, *Read_health_records*, *Prescribe*, and *Read_Prescription*.

While the usefulness of safety as security objective is clear, the availability as security objective may be tricky. Figure 2.7 represent the core RBAC for a data visualization

application that displays plots about the COVID-19 pandemic. The application must always display the plots p_6 , which is accessible to the basic Guest role. The other plots, namely p_1, p_2, p_3, p_4 , and p_5 can be shown only to the staff of the research center according to the roles the user has. Suppose that the most privileged roles are constrained by the DMER constraint $\text{SS-DMER}(\{\text{Researcher}, \text{BoardMember}\}, 2)$. Suppose that Alex wants to visualize p_3 in addition to p_6 . This amounts to an UAQ query with $P_{lb} = \{p_3, p_6\}$ together with the maximal number of plots in $P_{ub} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, in order to maximize the amount of information conveyed to the user while complying with the authorization constraints. In principle, Alex could activate the roles *Board Member*, *Researcher*, and *Employee*. However, only one between *Board Member* and *Researcher* can be activated, due to the DMER constraints. The permission p_3 could be activated by both the roles. The possible role solutions to the access query are:

- the activation of *Board Member* and *Employee*: it implies the activation of the permissions p_1, p_3, p_5 , and p_6 (4 permissions);
- the activation of *Researcher* and *Employee*: it implies the activation of the permissions p_2, p_3, p_4, p_5 , and p_6 (5 permissions);

Since the security objective is the data availability, the second solution is selected.

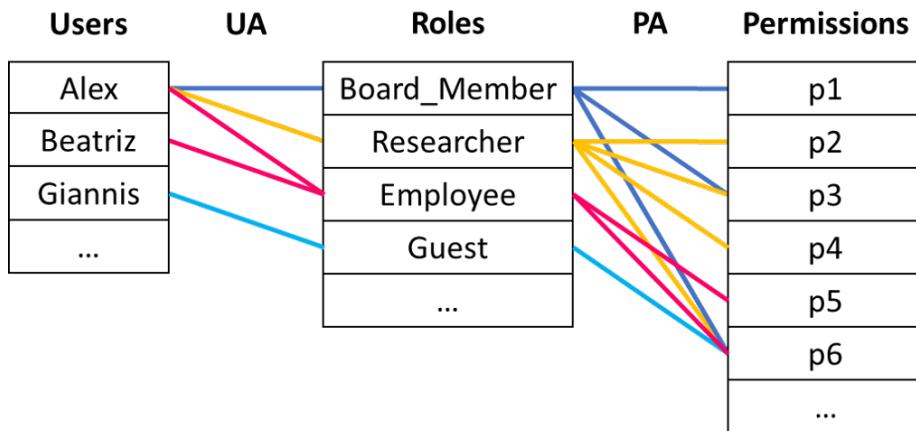


Figure 2.7 Representation of Core RBAC for the use case

In spite of its computational intractability (the UAQ problem has been shown to be **NP-hard** (28; 35)), several solvers based on a variety of techniques have been put forward along with encouraging experimental results. By borrowing ideas from constraint satisfaction, (90) puts forward two approaches to solving the UAQ problem: an extension of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm and a reduction to the Partial Maximum Satisfiability

Problem PMAX-SAT. The PMAX-SAT problem asks for the maximum number of clauses that can be satisfied by any assignment of a given subset of clauses (i.e., soft clauses) while the remaining clauses (i.e. hard clauses) must be satisfied. Both (13) and (64) improve the reduction proposed in (90) by using an efficient encoding of the cardinality constraints. However, while (13) advocates the use of state-of-the-art PMaxSAT solvers to solve a UAQ problem instance, (64) focuses on the decision version of the UAQ problem, which is then reduced to SAT. The latter proposes an off-the-shelf binary search algorithm to tackle the optimization problem by iteratively solving the respective decision version. The work of (64) also proves that the UAQ is fixed-parameter polynomial in the number of requested permissions, $|P_{lb}|$, (under a reasonable assumption) and provides an alternative procedure based on it. The result is of practical relevance: in fact, it applies to the UAQ problems seeking to minimize the number of granted permissions and states that the UAQ problems in this class can be solved in polynomial time if $|P_{ub}|$ is bounded by some positive integer regardless of the (maximum) size of the role set containing all permissions in P_{ub} , in symbols $|R_P|$.

Chapter 3

Related work

RBAC is a standard solution for access control. In the last two decades, it has been widely adopted by different commercial products. For this reason, RBAC and the UAQ problem for RBAC attracted the attention of many researchers. Besides, huge literature and community support do exist. In Section 3.1, we present the related work on the UAQ problem, in particular on its definition and computational complexity.

Many RBAC variants have been proposed with different expressive features, such as temporal constraints on role activations (36), spatial constraints (57; 58), and spatio-temporal constraints (85). The literature on constraints other than SMER and DMER constraints in the context of RBAC is huge, see for example (4; 5; 31; 49; 78; 87). Most of the constraints proposed are variants of SMER and DMER constraints. For example, in order to enforce SoD policies, permissions could be declared mutually exclusive instead of roles. In this way, a user cannot simultaneously be authorized for two or more mutually exclusive permissions. Even users could be mutually exclusive, so that they cannot be assigned to the same role.

However, because they are the fundamental types of constraints, in this thesis and in Section 3.2 we focus on the enforcement of Mutually Exclusive Roles constraints.

3.1 The UAQ problem: definition and complexity

UAQ is a central problem in RBAC systems; in fact, there is an increasing interest in developing techniques for tackling it efficiently and understanding its underlying complexity (53; 63; 86). To our knowledge, UAQ was first posed by Du and Joshi in (35), where the authors showed that the subcase of finding minimal set of roles to be activated in a session to address the permissions requested by the user is NP-hard, while the corresponding decision version (namely, without any optimization objective) is NP. The aforementioned

work analyzes UAQ in the presence of complex role hierarchies, however it does not consider the constraint types available in RBAC (e.g., mutual exclusion of roles) or the optimization of extra permissions. Then, Zhang and Joshi (93) generalized the UAQ problem by introducing the concept of optimization of the number of extra permissions in addition to the number of roles, thus dealing with availability and least-privilege. In addition, this work also introduces the constraints.

Besides, in the same work, the authors presented two algorithms for solving UAQ problem instances. The first algorithm is a two-step greedy search algorithm. In the first step, it selects a set of roles covering the requested permissions while trying to minimize the additional permissions these roles provide. This first step does not consider the constraints. In the second step, the algorithm checks whether the roles selected in the first step satisfy all the constraints. In the opposite case, the algorithm denies the user's request. It is efficient but incomplete: in fact, it is very likely that, in the first step, the algorithm chooses a set of roles violating some constraint. Consequently, it may deny the request even when there exists a set of roles that both satisfies the constraints and covers the desired permissions. The second algorithm aims to provide completeness. It is based on a simple generate-and test strategy and enumerates all subsets of roles assigned to the user until one is found that provides the needed permissions and satisfies all constraints. The approach to dealing with the unsoundness renders the algorithm inefficient: in fact, in the worst-case, this naive brute-force approach can be asked to generate 2^n solutions, where n is the number of roles assigned to the user.

Then Wickramaarachchi et al. extended the UAQ problem definition to a more generic form where the input includes both a lower bound and an upper bound of the requested permissions: solving the UAQ problem means to provide a set of roles that have permissions between the lower bound and upper bound, while satisfying all constraints (90). In addition, the authors propose a combination of mixing the optimization of number of roles and permissions different from the one presented in (93). In particular, the authors consider two possible optimization objectives: one is to prefer a set of roles that has permissions as close to the lower bound as possible, and the other is to prefer a set that have permissions as close to the upper bound as possible. The choice between the two optimization objectives depends on the nature and objective of the request, respectively safety or availability. By the way, a certain degree of safety is guaranteed even when the objective is availability: in fact, no permission not included in P_{ub} can be activated. Besides, the authors consider the exact match case, in which the lower bound and the upper bound coincide. Wickramaarachchi et al. also proposed two approaches for mitigating the intractability of UAQ, both leveraging prior work on boolean satisfiability. The first approach is an extension of the Davis-Putnam-Logemann-

Loveland (DPLL) algorithm (55), a backtracking based search algorithm used in SAT solving. This approach is exponential-time in its design. In addition, it does not address the joint optimization, namely the optimization of both the permissions and roles activation. In the second approach, the UAQ problem is reduced to the MaxSAT problem. After the reduction, an optimized off-the-shelf SAT solver can be used to solve the problem. The problem of deciding whether a boolean expression in CNF is satisfiable is known to be NP-complete (43). The authors' empirical observations show that the first approach is more efficient when an exact matching between the requested permissions and the available roles is sought. However, in general, the second solution performs better when $obj = \max$ or $obj = \min$ in the UAQ instance. Nevertheless, as pointed out in (63), the second solution is unsound and limited in the manner in which joint optimization is addressed. By the way, as reported in (13) and (63), both approaches show very poor performance when the number of roles increases and only single session DMER constraints are supported in authorization queries.

In (28), Chen and Crampton analyze the UAQ problem complexity for the cases of $obj = \max$ and $obj = \min$. The authors reduce the problem to special cases of set covering problems¹ (6; 43) for a given set of permissions P and roles R such that $Prms(r) \subseteq P$ holds for each $r \in R$, where $Prms(r)$ is the set of permissions assigned to r , and permission sets $P_{lb} \subseteq P_{ub} \subseteq P$ by using the following concepts:

- $\text{kernel}(P_{ub})$ denotes the largest set of permissions that are obtained from the set of roles $S \subseteq R$ such that $Prms(S) \subseteq P_{ub}$.
- the minimal subset (container) of roles such that the set of permissions obtained from the union of roles subsumes P_{lb} and there is no such other smaller subset of roles.

By using these concepts, when $obj = \max$ the optimum solution Q for a requested set of permissions P_{lb} is addressed by setting $Q = \text{kernel}(P_{ub})$ and is shown to be polynomial. However, this work does not consider the MER constraints, whose features are the main contributors to the UAQ problem complexity for $obj = \max$, as shown in (63). The case $obj = \min$ is addressed by restricting the whole set of permissions to $P = P_{ub}$ and finding the minimal subset of roles covering P_{lb} . This case is shown to be NP-hard.

In (64), Mousavi and Tripunitara distinguish between two versions of UAQ with constraints: decision and optimization. The former reduces a UAQ problem to SAT. The latter invokes a SAT solver iteratively with additional optimization objectives that minimize or

¹ Let $\mathbf{X} = \{1, 2, \dots, n\}$ be a ground set of n elements, and let \mathbf{S} be a family of subsets of \mathbf{X} , $|\mathbf{S}| = m$. A cover is a collection of sets such that their union is \mathbf{X} . Each $S \in \mathbf{S}$ has a non-negative cost $c(S)$ associated with it. The set covering problem is the problem of finding a cover of minimum cost. (6).

maximize the set of roles that can be activated. The authors show that the UAQ optimization problem is (fixed parameter) polynomial in $|P_{ub}|$ when the set of roles that can be activated is bounded by $|P_{ub}|$. The authors also show that the complexity result presented in (28) for the case $obj = \max$ changes when there are constraints in the UAQ instance. More specifically, they show that there is an upper bound (NP) for the general UAQ problem and the case $obj = \max$ also becomes intractable in UAQ instances with constraints.

To our knowledge, the last work on interest is presented in (47), where Lu et al. propose a comprehensive definition of the weighted UAQ (WUAQ) problem that considers the different nature and importance of each permission. The authors investigate the computational complexity of different subcases of the WUAQ problem and show that many instances in each subcase are intractable. Besides, they propose an algorithm to approximately solve the intractable cases of the WUAQ problem: this algorithm can be efficiently modified to handle the other subcases of the WUAQ problems.

3.2 Constraints enforcement

To our knowledge, the concept of Separation of Duties first appears in the information security literature in (73) under the name “separation-of-privilege”. Sandhu and Jajodia define SoD as a “timed-honored principle for prevention of fraud and errors, going back to the very beginning of commerce” (77). In (29), Clark and Wilson explain the need for enforcing SoD policies: “Because computers do not normally have direct sensors to monitor the real world, computers cannot directly verify external consistency. Rather, the correspondence is ensured indirectly by separating all operations into several subparts and requiring that each subpart be executed by a different person”. As Sandhu and Jajodia state in (77), “no single individual should be in a position to misappropriate assets on his own. Operationally, this means that a chain of events which affects the balance of assets must require different individuals to be involved at key points, so that without their collusion the overall chain cannot take effect”.

At a high level, a SoD policy can be enforced during *role assignment* (Static SoD, SSD), *role activation* (Dynamic SoD, DSD), or both. The difference between dynamic and static enforcement of SoD policies is presented in (68). In the former, users are constrained a priori from performing certain steps, while in the latter a user may perform any step in a sensitive task provided that she does not perform another step in that task. In many situations, the flexibility characteristic of the DSD policies makes them preferable with respect to the rigid SSD policies. A SSD policy can be enforced without maintaining a history for every task instance, by carefully assigning permissions to users. Instead, in order to

enforce a DSD policy, the system needs to maintain a history of the actions performed by the users. Before a user performs a step on the task, the system must check if the SoD policy is not violated. Sandhu presented a history-based mechanism for dynamically enforcing SoD policies named transaction control expressions (74; 75). Foley proposed a framework based on relabel policies (80) to express dynamic SoD requirements (41).

Recent research has focused on addressing constraints and security design principles individually (27; 35) with run-time context (59). Chen and Crampton (27) try to support the least privilege principle as a role mapping problem between different domains. Du and Joshi (35) also focus on the least privilege in a distributed setting. The authors define the inter-domain role mapping (IDRM) problem, namely the problem of finding the set of roles in a domain that are authorized for a set of permissions requested from an external domain. However, role assignment may be driven with different business objectives and may result in an overly restrictive authorization system that makes enforcement during role activation more desirable.

A general constraint enforcement framework is presented in (32) in which the enforcement of constraints is transformed into an authorization checking problem by using RBAC states. They employ relational algebra to check whether a derived state (after a set of events) is among the permitted states (or prohibited states).

In (40), Ferrini and Bertino discuss the enforcement of SoD constraints in eXtensible Access Control Markup Language (XACML) (30) policies. The authors encode state information about the exercise of permissions in ontologies whose inconsistency refers to SoD violation. When a request is permitted, XACML obligations are used to update the state information.

Xu et al. (59) proposed to use XACML in administrative RBAC systems to specify how administrative changes in the policies can be reflected at run-time, by changing the role definitions and the user assignment. They investigated the use of lock mechanisms to handle concurrency of sessions and provided XACML solutions for the constraint enforcement.

Li and Wang present SoD algebra (SoDA) (52), a solution for the specification of SoD constraints. In addition to the formalism for constraint specification, the authors provide a methodology for secure workflow design. However, the work of Li and Wang does not exploit the notion of sessions in RBAC. The SoD enforcement can be achieved through static enforcement via static safety checking (SSC) or dynamic enforcement via term satisfiability (TSAT). Once defined a workflow, each step of the workflow is assigned to a userset regardless of the objects and the ownerships. Since it does not allow changes in authorizations at run-time, SoDA could be too restrictive in terms of permissions that can be exercised by a single

user. In order to enable tighter integration with the workflow systems, Basin et al. extend SoDA semantics to multisets and provide a mapping of SoDA terms to workflows specified in communicating sequential processes (CSP) expressions for dynamic enforcement of SoD constraints (18). In this way, they could define a secure workflow (SW) process as a parallel composition of RBAC (actual authorizations) and the business workflow itself (business events). All these efforts result in high level policies on top of actual RBAC policies that regulate the run-time permissions at a more abstract level. Even if this approach eases the policy specification, it could increase the complexity in run-time permission management.

The main motivations of these works are the inherent limitations of RBAC standard. In fact, the latter defines a set of high level functions to model the requirements of applications while providing a bird-eye view to authorization; however, the use of these functions requires careful consideration for correctly enforcing dynamic constraints and supporting well-known security design principles, such as least privilege. As Sandhu et al. stated in their paper that introduced the highly influential RBAC96 family of RBAC models (76), “The most common RBAC constraint is mutually exclusive roles. The same user can be assigned to at most one role in a mutually exclusive set. This supports separation of duties, which is further ensured by a mutual exclusion constraint on permission assignment”. In particular, MER constraints that prevent any user from being a member of mutually exclusive roles are named Static MER (SMER) constraints; instead, constraints that prevent any user from activating mutually exclusively roles simultaneously in a session are named Dynamic MER (DMER) constraints.

Within this context, sessions allow a finer control on the management of permissions, since they enable the role activation and de-activation. However, sessions are underspecified (51; 52) in the RBAC standard. In (51), Li et al. show that the RBAC standard lacks proper functionality in supporting SoD: in fact, even though the constraints are enforced in a single session, the RBAC standard does not prevent the user from activating mutually exclusive roles across multiple sessions, resulting in a SoD policy violation. However, this should not be surprising: in fact, DMER constraints are motivated by the least privilege principle rather than the SoD principle. The ANSI/NIST standard itself states that “DSD properties provide extended support for the principle of least privilege in that each user has different levels of permission at different times, depending on the task being performed” (46).

Li et al. show that even the enforcement of SSD constraints is intractable (coNP-complete), and proposed to convert SSD requirements into RBAC SMER (51). The solution proposed by Zhang and Joshi (93) and Wickramaarachchi et al. (90), already cited in Section 3.1, do the same with DMER constraints: they formulated the User Authorization Query (UAQ) problem including the definition of DMER constraints. However, both the approaches

considered only single sessions for authorization queries. More importantly, they have not included the history in decision making. As a consequence, the solutions are not adequate to enforce SoD policies. Armando et al. propose a solution to this problem in (13). Their work enables the use of existing SAT tools for constraint enforcement by grounding the problem of user authorization queries to model checking problem. In particular, they put "sessions" together with the history as core concepts in the model and introduce four different kinds of DMER constraints, namely:

- single-session dynamic MER constraint;
- multi-session dynamic MER constraint;
- single-session history-based MER constraint;
- multi-session history-based MER constraint.

The latter will be better discussed in Section 4.2. By the way, their definition enables to enforce SoD policies dynamically.

Chapter 4

UAQ problem formalization

4.1 Core RBAC

An *RBAC policy* is a tuple $RP = (U, R, P, UA, PA, \succeq, C)$, where U is a set of users, R a set of roles, and P a set of permissions; users are associated to roles by the user assignment relation $UA \subseteq U \times R$ and roles are associated to permissions by the permission assignment relation $PA \subseteq R \times P$; \succeq is a partial order on R , modeling the hierarchy between roles, i.e. $r_1 \succeq r_2$ means that r_1 is *more senior than* r_2 , namely $(r_2, p) \in PA \implies (r_1, p) \in PA$ for $r_1, r_2 \in R$ and $p \in P$; and C is a set of *dynamic mutually exclusive role (DMER)* constraints as defined earlier. In the example presented in Section 2.2.1:

- $U = \{Richard, Claire, Sarah, Matthias, Jane\}$;
- $R = \{Doctor, Data_Manager, Nurse, Patient, Head_Physician, Pharmacist\}$;
- $P = \{Read_id, Read_health_records, Prescribe, Send_data, Read_prescription, Manage_schedule, Check_process, Approve_dispensation\}$;
- $C = \{\text{SS-DMER}(\{Doctor, Data_Manager\}, 2)\}$.

Besides, Figure 2.3 shows the user assignment and the user assignment relations.

Let $u \in U$, we define $R_u = \{r' \in R : r \succeq r'\text{ for some }r \in R\text{ such that }(u, r) \in UA\}$. For example,

$$R_{Richard} = \{Doctor, Data_Manager\}.$$

A user u has permission p iff $(r, p) \in PA$ for some $r \in R_u$. We assume that RBAC policies considered are *finite*, i.e. U , R , P , and C have finite cardinality. Moreover, we treat

permissions as if they are opaque (i.e., we treat permissions as labels, not as operations on objects) and mutually independent, i.e. the possession of one or more permissions does not imply the possession of another permission. Let $p \in P$, we define $R_p = \{r' \in R : r' \succeq r \text{ for some } r \in R \text{ such that } (r, p) \in PA\}$ and $R_{P'} = \bigcup_{p \in P'} R_p$ for any $P' \subseteq P$. Similarly, if $r \in R$ then $P_r = \{p \in P : (r', p) \in PA \text{ for some } r' \in R \text{ such that } r \succeq r'\}$ and $P_{R'} = \bigcup_{r \in R'} P_r$ for any $R' \subseteq R$. For example:

- $R_{Read_prescription} = \{Doctor, Nurse, Patient\};$
- $R_{\{Send_data, Check_process\}} = \{Data_Manager, Head_Physician\};$
- $P_{Patient} = \{Read_health_records, Read_prescription\};$
- $P_{\{Nurse, Pharmacist\}} = \{Read_prescription, Approve_dispensation\}.$

4.2 Dynamic Mutually-Exclusive Roles Constraints

Let C be a finite set of DMER constraints of the form $SS\text{-DMER}(rs, t)$. Let $\rho \subseteq R$ be a set of roles active in a given session, then we say that ρ satisfies $c \in C$ iff

- $[SS\text{-DMER}(rs, t)]$ for all $s \in S$, $|rs \cap \rho| < t$, i.e. **for every session** the number of **simultaneously** active roles in rs is smaller than t .

We say that ρ satisfies C iff ρ satisfies c for all $c \in C$.

Let S be a set of sessions and $user : S \rightarrow U$ a function that associates each session $s \in S$ with the corresponding user.

To define DMER constraints that span multiple sessions and the role activation history of a user, we must define the state of the system as a triple (S, α, π) , where $\alpha : S \rightarrow 2^R$ associates each session $s \in S$ with the set of roles currently active in that session and $\pi : S \rightarrow 2^R$ associates each session $s \in S$ with the set of roles that have been active in that session in the past. Clearly, $\alpha(s) \subseteq R_{user(s)}$ and $\pi(s) \subseteq R_{user(s)}$.

Let C be a finite set of DMER constraints of the form $SS\text{-DMER}(rs, t)$, $MS\text{-DMER}(rs, t)$, $SS\text{-HMER}(rs, t)$ or $MS\text{-HMER}(rs, t)$. If $u \in U$, then S_u denotes the set of sessions associated with u , i.e. $S_u = \{s \in S : user(s) = u\}$. We say that (S, α, π) satisfies c iff

- $[SS\text{-DMER}(rs, t)]$ for all $s \in S$, $|rs \cap \alpha(s)| < t$, i.e. **for every session** the number of **simultaneously** active roles in rs is smaller than t .

- [MS-DMER(rs, t)] for all $u \in U$, $|rs \cap \bigcup_{s \in S_u} \alpha(s)| < t$, i.e. **for every user** the number of **simultaneously** active roles in rs is smaller than t .
- [SS-HMER(rs, t)] for all $s \in S$, $|rs \cap \pi(s)| < t$, i.e. **for every session** the number of roles in rs activated **over time** is smaller than t ;
- [MS-HMER(rs, t)] for all $u \in U$, $|rs \cap \bigcup_{s \in S_u} \pi(s)| < t$, i.e. **for every user** the number of roles in rs activated **over time** is smaller than t .

We say that (S, α, π) *satisfies* C iff (S, α, π) satisfies c for all $c \in C$. If (S, α, π) satisfies C , then we say that (S, α, π) is a *valid* state.

Referring to the running example introduced in Section 2.2.1, Table 4.1 schematically represents the differences among the following DMER constraints:

- SS-DMER($\{\text{Doctor}, \text{Data_Manager}\}$, 2);
- MS-DMER($\{\text{Doctor}, \text{Data_Manager}\}$, 2);
- SS-HMER($\{\text{Doctor}, \text{Data_Manager}\}$, 2);
- MS-HMER($\{\text{Doctor}, \text{Data_Manager}\}$, 2).

In particular, the table shows whether each constraint allows (\checkmark) or forbid (\times) the presented lists of actions performed in sessions s_1 and s_2 . In Section 4.4 we will show how MS-DMER, SS-HMER and MS-HMER can be reduced to traditional SS-DMER constraints.

Table 4.1 Actions allowed (\checkmark) or forbidden (\times) by the different kinds of DMER constraints with $rs = \{Doctor, Data_Manager\}$ and $t = 2$

Actions	SS-DMER	MS-DMER	SS-HMER	MS-HMER
1. Activate Doctor in s_1	\times	\times	\times	\times
2. Activate Data_Manager in s_1				
1. Activate Doctor in s_1	\checkmark	\times	\checkmark	\times
2. Activate Data_Manager in s_2				
1. Activate Doctor in s_1				
2. Deactivate Doctor in s_1	\checkmark	\checkmark	\times	\times
3. Activate Data_Manager in s_1				
1. Activate Doctor in s_1				
2. Deactivate Doctor in s_1	\checkmark	\checkmark	\checkmark	\times
3. Activate Data_Manager in s_2				

4.3 The User Authorization Query

A *User Authorization Query (UAQ)* is a tuple $q = (s, P_{lb}, P_{ub}, o_p)$, where $s \in S$, $P_{lb} \subseteq P_{ub} \subseteq P$, and $o_p \in \{any, min, max\}$ is the permission optimization objective .

Definition 4.3.1 (UAQ problem) The UAQ problem for $q = (s, P_{lb}, P_{ub}, o_p)$ in RP is the problem of determining a set of roles $\rho \subseteq R_{user(s)}$ such that (64):

1. ρ satisfies C,
2. $P_{lb} \subseteq P_\rho \subseteq P_{ub}$, and
3. any other $\rho' \subseteq R_{user(s)}$ that satisfies C and $P_{lb} \subseteq P_{\rho'} \subseteq P_{ub}$ is such that:
 - $P_\rho \subseteq P_{\rho'}$, if $o_p = min$;
 - $P_{\rho'} \subseteq P_\rho$, if $o_p = max$.

For example, we refer to the UAQ policy presented in Figure 2.3 and, in addition, we define the following constraint:

$$C = \{\text{SS-DMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)\}.$$

Suppose that Matthias needs the permission *Check_process*, while the permission *Send_data* must not be activated to avoid an unnecessary risk. Suppose also that $o_p = \text{min}$. The resulting query is:

```

$$\begin{aligned} q = ( \\ & s = s_1, \\ & P_{lb} = \{\text{Check\_process}\}, \\ & P_{ub} = \{\text{Read\_id}, \text{Read\_health\_records}, \text{Prescribe}, \text{Read\_prescription}, \\ & \quad \text{Manage\_schedule}, \text{Check\_process}\}, \\ & o_p = \text{min} \\ ) \end{aligned}$$

```

As depicted in Figure 4.1, Matthias can only activate the roles *Doctor*, *Data_Manager* and *Head_Physician*. As a consequence, he can not activate the permission *Approve_dispensation*. The requested permission *Check_process* is assigned only to the role *Head_Physician*, then this role must appear in the solution. Matthias can not activate *Data_Manager*, because the permission *Send_data* assigned to it is not contained in $|P_{ub}|$. In addition, he can not concurrently activate the roles *Doctor* and *Data_Manager*, due to the DMER constraint. As a consequence, the only possible role activations are:

1. $\rho'_1 = \{\text{Head_Physician}\}$: this role activation implies that $P_{\rho'} = \{\text{Check_process}, \text{Manage_schedule}\}$. Then, ρ'_1 activates 1 extra permission;
2. $\rho'_2 = \{\text{Head_Physician}, \text{Doctor}\}$: this role activation implies that $P_{\rho'} = \{\text{Check_process}, \text{Manage_schedule}, \text{Read_id}, \text{Read_health_records}, \text{Prescribe}, \text{Read_prescription}\}$. Then, ρ'_2 activates 4 extra permissions.

Since $o_p = \text{min}$, it follows that the solution to the UAQ query is $\rho = \rho'_1$. It is then clear that, in case of $o_p = \text{max}$, the solution would have been $\rho = \rho'_2$.

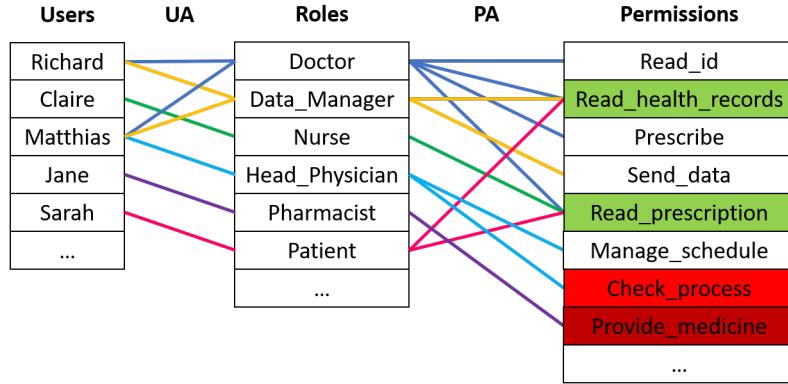


Figure 4.1 Representation of the example

Definition 4.3.2 (UAQ problem – multi-session and history-based DMER) Let (S, α, π) be a valid state. The UAQ problem for $q = (s, P_{lb}, P_{ub}, o_p)$ in RP and (S, α, π) is the problem of determining a set of roles $\rho \subseteq R_{user(s)}$ such that:

1. $(S, \alpha[s \leftarrow \rho], \pi[s \leftarrow \rho])$ satisfies C, where $\alpha[s \leftarrow \rho]$ and $\pi[s \leftarrow \rho]$ indicates respectively the set of roles that would be active and the set of roles the would have been active in the past in case ρ is taken as solution,
2. $P_{lb} \subseteq P_\rho \subseteq P_{ub}$, and
3. any other $\rho' \subseteq R_{user(s)}$ such that $(S, \alpha[s \leftarrow \rho'], \pi[s \leftarrow \rho'])$ satisfies C and $P_{lb} \subseteq P_{\rho'} \subseteq P_{ub}$ is such that:
 - $P_\rho \subseteq P_{\rho'}$, if $o_p = min$;
 - $P_{\rho'} \subseteq P_\rho$, if $o_p = max$.

Our definition of UAQ problem extends the one given in (64) by supporting multi-session and history-based DMER. Notice that the case $o_p = any$ can be used for addressing the “exact match” case of (93) by setting $P_{lb} = P_{ub}$ as shown in (90).

The UAQ problem can be further extended to support the *joint optimization* of roles and permission (64). This can be done by adding to the UAQ query an optimization objective for roles (o_r) and a priority (pri) to indicate which of the two optimization objectives must have the precedence when they are both different from *any*. This extension results in Definition 4.3.3.

Definition 4.3.3 (UAQ problem – permission-based and joint optimization) Let (S, α, π) be a valid state. The UAQ problem for $q = (s, P_{lb}, P_{ub}, o_p, o_r, pri)$ in RP and (S, α, π) is the problem of determining a set of roles $\rho \subseteq R_{user(s)}$ such that:

1. $(S, \alpha[s \leftarrow \rho], \pi[s \leftarrow \rho])$ satisfies C ,
2. $P_{lb} \subseteq P_\rho \subseteq P_{ub}$, and
3. any other $\rho' \subseteq R_{user(s)}$ such that $(S, \alpha[s \leftarrow \rho'], \pi[s \leftarrow \rho'])$ satisfies C and $P_{lb} \subseteq P_{\rho'} \subseteq P_{ub}$ is such that:
 - if $pri = p$:
 - $P_\rho \subseteq P_{\rho'}$, if $o_p = min$;
 - $P_{\rho'} \subseteq P_\rho$, if $o_p = max$,
 then, if o_r is not any,
 - $\rho \subseteq \rho'$, if $o_r = min$;
 - $\rho' \subseteq \rho$, if $o_r = max$;
 - if $pri = r$:
 - $\rho \subseteq \rho'$, if $o_r = min$;
 - $\rho' \subseteq \rho$, if $o_r = max$,
 then, if o_p is not any,
 - $P_\rho \subseteq P_{\rho'}$, if $o_p = min$;
 - $P_{\rho'} \subseteq P_\rho$, if $o_p = max$.

As shown by Definitions 4.3.1 and 4.3.3, different kinds of optimizations do exist:

- *permission-based optimization*: $pri = p$ and the role objective is *any*;
- *role-based optimization*: $pri = r$ and the permission objective is *any*;
- *joint optimization*: neither o_p or o_r is *any*.

For the sake of completeness, we also provide an example with joint optimization. In particular, we consider the example provided before, but with $o_p = min$, $o_r = max$ and $pri = r$. The resulting UAQ query is:

$$q = ($$

$$s = s_1,$$

$$P_{lb} = \{Check_process\},$$

$$P_{ub} = \{Read_id, Read_health_records, Prescribe, Read_prescription,$$

$$Manage_schedule, Check_process\},$$

$$o_p = min$$

```


$$o_r = \max$$


$$pri = r$$


$$)$$


```

Since the main objective is to maximize the activation of roles, we first search for solutions satisfying this target. If more solutions do exist, then we choose the one that minimizes the number of extra permissions. The possible role activations are the same as in the previous example:

1. $\rho'_1 = \{\text{Head_Physician}\}$: ρ'_1 activates only 1 role;
2. $\rho'_2 = \{\text{Head_Physician}, \text{Doctor}\}$: ρ'_2 activates 2 roles.

The solution of the UAQ query is then $\rho = \rho'_2$. In this example, ρ'_2 would be the solution also in case of role-based optimization.

In our work, we focus on simple permission optimization, namely

- $pri = p$,
- $o_p = \{\min, \max\}$, and
- $o_r = \{\text{any}\}$

and joint optimization with permission optimization as priority, namely

- $pri = p$,
- $o_p = \{\min, \max\}$, and
- $o_r = \{\min, \max\}$.

However, the methodology presented in the next sections can be easily extended and employed also in case of joint optimization with $pri = r$.

4.4 Reduction to SS-DMER constraints

As a first contribution, we demonstrate how SS-HMER, MS-DMER, and MS-HMER constraints can be reduced to SS-DMER constraints.

Theorem 4.4.1 *Let RP be an RBAC policy, $\sigma = (S, \alpha, \pi)$ be a valid state for RP and q is a query for session s. It can be shown that ρ is a solution to q in RP and $\sigma = (S, \alpha, \pi)$ iff ρ is a solution to q in RP' , where RP' is obtained from RP by replacing the sets of constraints C with $C[s, \sigma]$, where $C[s, \sigma]$ is a set of constraints in which MS-DMER, SS-HMER and MS-HMER for the particular session s are replaced with SS-DMER constraints whose rs and t varies with the state σ .*

The reader can find the proof of Theorem 4.4.1 in Appendix A.

In practice, Theorem 4.4.1 states that, given the role activation history, any extended DMER can be reduced to a traditional SS-DMER whose rs and t depend on the action performed by the user. To bring a fuller appreciation of the theorem, we show an application on the example introduced in Section 2.2.1. To this aim, we consider the user Richard and the constraint $MS\text{-}DMER(\{Doctor, Data_Manager\}, 2)$. We recall that the role *Doctor* activates the permissions *Read_id*, *Read_health_records*, *Prescribe* and *Read_prescription*, while *Data_Manager* activates *Read_health_records* and *Send_data*. In Table 4.2, for any step, we present:

- the open sessions and the active roles;
- the query performed by Richard;
- the SS-DMER resulting from the reduction;
- the outcome of the query.

Below, we describe the actions performed by Richard.

Step 0: no session is open;

Step 1: Richard creates session s_1 ;

Step 2: Richard wants to access Sarah's health records. As a consequence, he needs the permissions *Read_id* and *Read_health_record* in s_1 . Since no session other than s_1 is open, the original MS-DMER is reduced to $SS\text{-}DMER(\{Doctor, Data_Manager\}, 2)$ in s_1 : namely, one (and only one) between the roles *Doctor* and *Data_Manager* can be activated. *Doctor* is then activated in s_1 to provide Richard with the permissions needed to perform his task;

Step 3: the role *Doctor* is active in session s_1 . Richard can access Sarah's health records;

Step 4: Richard creates session s_2 . *Doctor* is still active in s_1 ;

Step 5: Richard wants to send data to the pharmaceutical company in session s_2 . To this aim, he needs the permissions *Read_health_records* and *Send_data*. However, the role *Doctor* is active in session s_1 : for this reason, to satisfy the original MS-DMER, the latter is reduced to the constraint $\text{SS-DMER}(\{\text{Data_Manager}\}, 1)$ in s_2 : the latter actually means that the role *Data_Manager* cannot be activated. As a result, the requested access is denied;

Step 6: Richard closes s_1 . Only session s_2 is open, but no role is active;

Step 7: Richard tries again to send data to the pharmaceutical company in session s_2 . The same query as in Step 5 is performed. This time, neither *Doctor* or *Data_Manager* is active in any active session. As a consequence, in order to enforce the MS-DMER, it is enough to impose the constraint $\text{SS-DMER}(\{\text{Doctor, Data_Manager}\}, 2)$. The role *Data_Manager* can then be activated to provide the user with the requested permissions, while *Doctor* must not be activated;

Step 8: the role *Data_Manager* is active in s_2 . Richard can then send data to the pharmaceutical company in session s_2 ;

Step 9: Richard closes s_2 . No session is active.

Table 4.2 Application of Theorem 4.4.1 on the running example introduced in Section 2.2.1

Step	Sessions (active roles)	Query	DMER	Answer
0	-			
1	$s_1()$			
2	$s_1()$	<i>Read_id</i> and <i>Read_health_records</i> in s_1	SS-DMER($\{Doctor, Data_Manager\}, 2$)	Allow
3	$s_1(Doctor)$			
4	$s_1(Doctor)$ $s_2()$			
5	$s_1(Doctor)$ $s_2()$	<i>Read_health_records</i> and <i>Send_data</i> in s_2	SS-DMER($\{Data_Manager\}, 1$)	Deny
6	$s_2()$			
7	$s_2()$	<i>Read_health_records</i> and <i>Send_data</i> in s_2	SS-DMER($\{Doctor, Data_Manager\}, 2$)	Allow
8	s_2 (<i>Data_Manager</i>)			
9	-			

Chapter 5

Solving the UAQ problem

The literature offers a variety of algorithms with different complexities to solve the UAQ problem. The algorithms can be summarized in two categories: algorithms that employ variants of search techniques and algorithms that reduce the UAQ problem to (variants of) SAT. In this section, we provide a systematic overview of these algorithms, also giving insights into the complexity of the UAQ problem.

For the sake of simplicity but without loss of generality, we assume that:

- no role in R is assigned permissions that are not contained in P_{ub} , and
- $P_{ub} = P$.

In fact, all the permissions in $P \setminus P_{ub}$ must not be granted; therefore all the roles that activate these permissions *cannot be included in a solution*. These roles and the permissions in $P \setminus P_{ub}$ can thus be safely removed from the policy (in polynomial time). By doing that, we simply exclude UAQ problems that are de facto equivalent to UAQ problems that meet our assumptions.

5.1 Search-based techniques

Search-based algorithms tackle the UAQ problem directly without further reduction. Perhaps the earliest, (93) presents a naive two-step algorithm that first obtains a set of roles covering the desired permissions minimally through a greedy search. Then the algorithm checks whether the set of roles satisfies the constraints. This algorithm is not sound since the first step does not consider any constraints and thus may not find a combination of roles that satisfies them. Discussed again in (93) and improved in (90), an alternative sound approach to solving a UAQ instance amounts to:

Algorithm 1

1. enumerate all possible role activations for the user,
2. check (in polynomial time as shown in (64)) whether the selected roles grant the requested permissions (i.e., fall between P_{lb} and P_{ub}), and satisfy the DMER constraints, and
3. keep the best (according to the security objective considered) solution encountered, if any.

The algorithm is clearly in $O(2^{|R|})$. The DPPL-based procedures proposed in (90) and the DFS-based algorithms proposed in (53) are optimized versions of this algorithm with additional preprocessing and pruning steps.

An alternative approach to solving the UAQ problem (adapted from (64)) is as follows:

Algorithm 2

1. enumerate all sets of roles $S_p \subseteq R_p$ for each $p \in P_{ub}$,
2. check in polynomial time whether $S = \bigcup_{p \in P_{ub}} S_p$ is such that $P_{lb} \subseteq P_S \subseteq P_{ub}$ and S satisfies the DMER constraints, and
3. keep the best (according to the security objective considered) solution encountered, if any.

The above algorithm is in $O(2^{\widehat{R}_P |P_{ub}|})$, where $\widehat{R}_P = \max_{p \in P} |R_p|$. To see this, it suffices to observe that for each $p \in P_{ub}$ there are at most $2^{|R_p|}$ subsets S_p of R_p , and thus there are in total at most $2^{\widehat{R}_P |P_{ub}|}$ candidate solutions to consider. When it is sufficient to activate “at most one role per permission”, (64) shows that the above algorithm is in $O(\widehat{R}_P^{|P_{ub}|})$ and hence is also fixed-parameter polynomial (FPP) in $|P_{ub}|$, i.e., it is polynomial-time if $|P_{ub}| \leq c$ for some constant c .

If the objective is min or any, then it is sufficient to activate at most one role per permission. This leads to the following, more efficient version of the algorithm:

Algorithm 3

1. enumerate all roles $r_p \in R_p$ for each $p \in P_{lb}$,
2. check in polynomial time whether $S = \{r_p \in R_p : p \in P_{lb}\}$ is such that $P_S \subseteq P_{ub}$ and S satisfies the DMER constraints, and
3. keep the best (according to the security objective considered) solution encountered, if any.

If the objective is min or any, it is in fact possible to consider the activation of individual roles granting the permissions in P_{lb} and the algorithm is in $O(|\widehat{R}_P|^{P_{lb}})$ (64).

If the optimization objective is max, then the “at most one role per permission” assumption does not hold and hence the FPP result cannot be applied in general. To illustrate consider a UAQ problem with $R = \{r_1, r_2, r_3\}$, $P = \{p_1, p_2, p_3, p_4\}$, PA such that $P_{r_1} = \{p_1, p_3\}$, $P_{r_2} = \{p_2, p_4\}$, $P_{r_3} = \{p_2, p_3\}$, $P_{lb} = \{p_1\}$, $P_{ub} = P$ and $obj = \text{max}$. The solution $\{r_1\}$ satisfies the “at most one role per permission” assumption but it is not optimal. In fact both $\{r_1, r_2\}$ and $\{r_1, r_2, r_3\}$ activate a larger (actually maximal) set of permissions, namely P . An alternative approach to tackling the problem for the max case is put forward in (64):

Algorithm 4

1. enumerate all sets of possible role activations $R_a = R_1 \cup \dots \cup R_{|C|} \cup R_{free}$, where $R_i \subseteq rs_i$ and $|R_i| < t_i$, for all constraints $DMER(rs_i, t_i)$ in C and $i = 1, \dots, |C|$, and $R_{free} \subseteq R$ is the set roles that do not occur in the DMER constraints (and can thus be freely activated),
2. check in polynomial time whether $P_{lb} \subseteq P_{R_a} \subseteq P_{ub}$ and R_a satisfies the DMER constraints.
3. keep the maximum sized $|P_{R_a}|$ solution encountered, if any.

We now note that for each constraint $DMER(rs_i, t_i)$ in C there are $\sum_{k=1}^t \binom{|rs_i|}{k}$ subsets R_i of rs_i such that $|R_i| < t_i$. The number of sets $R_1 \cup \dots \cup R_n$ is at most $\left(\sum_{k=1}^t \binom{|rs_i|}{k}\right)^{|C|}$,¹

¹It is equal to $\left(\sum_{k=1}^t \binom{|rs_i|}{k}\right)^{|C|}$ only if rs_i and rs_j are pairwise disjoint for all $i, j = 1, \dots, |C|$.

from which it easily follows (by using the well-known upper bound $\sum_{k=0}^t \binom{n}{k} \leq (1+n)^t$, that the enumeration of the set of roles $R_a = R_1 \cup \dots \cup R_{|C|} \cup R_{free}$ grows as $O(\widehat{rs}^{|C|\widehat{t}})$, where $\widehat{rs} = \max_{DMER(rs,t) \in C} |rs|$ and $\widehat{t} = \max_{DMER(rs,t) \in C} t$. This improves the upper bound $O(|R|^{|C|\widehat{t}})$ given in (64). As shown in (64), the role hierarchy, \succeq , does not contribute to the computational complexity of the problem.

Table 5.1 provides a summary of the algorithmic complexity results.

Table 5.1 Complexity of search-based techniques

Algorithm	Objective	Complexity
1	any, min, max	$O(2^{ R })$
2	any, min, max	$O(2^{\widehat{R}_P} P_{ub})$
3	any, min	$O(\widehat{R}_P^{ P_{lb} })$
4	max	$O(\widehat{rs}^{ C \widehat{t}})$

It is interesting to note that, even if the algorithms presented in this section are meant to solve permission-based optimization problems, they are also adequate to solve joint optimization problems with $pri = p$. In fact, it is enough to change the definition of “best solution encountered”, that, in case of joint optimization, takes o_r into account after the permission optimization.

5.2 SAT-based techniques

Let $RP = (U, R, P, UA, PA, \succeq, C)$ be an RBAC policy where C is a set of constraints of the form $SS\text{-}DMER(rs, t)$ and $q = (s, P_{lb}, P_{ub}, o_r, o_p, pri)$ a UAQ query for RP . Since RP is finite (i.e. the set U of users, R of roles, and P of permissions are all finite), the UAQ problem can be reduced to (variants of) the Boolean Satisfiability Problem (SAT), that can be solved by available state-of-the-art SAT solvers.

A Boolean expression (also called propositional logic formula) is a formula built with Boolean variables, *and* (\wedge), *or* (\vee), *not* operators (\neg), and parentheses. The Boolean satisfiability problem is the problem of determining if there exists a truth-value assignment to the variables that satisfies the Boolean formula. In other words, it is the problem of determining if it is possible to replace the variables with a logical value (*true* or *false*) so that the Boolean expression evaluates to *true*. If so, the formula is satisfiable, otherwise it is unsatisfiable. In the Boolean formula, the variable which is negated through a not operator is

said negative literal. In the opposite case, the variable can be referred to as positive literal. A clause is a disjunction of literals (namely a formula consisting of *or* of literals), or a single literal. For example,

$$a \vee \neg b$$

is a clause, in which a is a positive literal and $\neg b$ is a negative literal. A Boolean formula is said to be in Conjunctive Normal Form (CNF) if it is a conjunction of clauses (namely if it is an *and* of clauses) or a single clause. For example,

$$(a \vee \neg b) \wedge (\neg a \vee \neg b)$$

is a Boolean formula in CNF consisting of two clauses, namely $a \vee \neg b$ and $\neg a \vee \neg b$. In particular it is satisfiable, in fact, for example, with the truth-assignment $\{a = \text{true}, b = \text{false}\}$ the formula evaluates to true:

$$\begin{aligned} (a \vee \neg b) \wedge (\neg a \vee \neg b) &= \\ &= (\text{true} \vee \neg \text{false}) \wedge (\neg \text{true} \vee \neg \text{false}) = \\ &= \text{true} \wedge \text{true} = \\ &= \text{true} \end{aligned}$$

SAT solvers can be used to tackle the UAQ problem in a variety of ways. A first approach (64) amounts to reducing the UAQ Decision Problem (i.e., a variant of the UAQ problem whereby the optimization objective is replaced by setting a bound that must be met by the solution) to SAT and solving the optimization problem through binary search that leverages the SAT solver as an oracle for the decision problem. A second approach (13; 86; 90) eliminates the need for the binary search by directly encoding UAQ problems into PMaxSAT. UAQ-Solve leverages a reduction to the Weighted Partial MAX-SAT (WPMaxSAT) problem.

A *Weighted Partial MAX-SAT problem* is a triple $(\mathcal{H}, \mathcal{S}, w)$, where \mathcal{H} and \mathcal{S} are two sets of propositional clauses, called “hard” and “soft” respectively, and w is a function that associates a positive integer to each clause in \mathcal{S} . A *solution to a Weighted Partial MAX-SAT problem* $(\mathcal{H}, \mathcal{S}, w)$ is a truth-value assignment to the propositional variables in \mathcal{H} and \mathcal{S} that satisfies all clauses in \mathcal{H} and minimizes the solution cost, namely the sum of the weights of unsatisfied clauses in \mathcal{S} .

We assume the existence of a propositional variable \bar{r} for each $r \in R$ and a propositional variable \bar{p} for each $p \in P$. We define \mathcal{C}_{RP} as the smallest set containing the propositional

clauses presented in the paragraphs below. In order to supply the reader with examples, we refer to the UAQ policy presented in Figure 2.3 and additionally

$$C = \{\text{SS-DMER}(\{\text{Doctor}, \text{Data_Manager}\}, 2)\}.$$

Suppose also that Matthias needs the permissions

Read_health_records and *Read_prescription*, while the permission *Check_process* must not be activated. Finally, suppose that permission-based optimization is used and that $o_p = \text{min}$. The aforementioned conditions result in the following query:

```

 $q = ($ 
     $s = s_1,$ 
     $P_{lb} = \{\text{Read\_health\_records}, \text{Read\_prescription}\},$ 
     $P_{ub} = \{\text{Read\_id}, \text{Read\_health\_records}, \text{Prescribe}, \text{Send\_data},$ 
         $\text{Read\_prescription}, \text{Manage\_schedule}\},$ 
     $o_r = \text{any},$ 
     $o_p = \text{min},$ 
     $pri = p$ 
 $)$ 

```

Core RBAC

In the Core RBAC, \mathcal{C}_{RP} contains the following hard clauses:

1. $\neg\bar{r}$ for all $r \in R$ such that $(\text{user}(s), r) \notin UA$. In our example:

- $\neg\overline{\text{Nurse}}$;
- $\neg\overline{\text{Pharmacist}}$;
- $\neg\overline{\text{Patient}}$;

2. $(\neg\bar{r} \vee \bar{p})$ for all $p \in P$ and $r \in R_p$. In our example:

- $\neg\overline{\text{Doctor}} \vee \overline{\text{Read_id}}$;
- $\neg\overline{\text{Doctor}} \vee \overline{\text{Read_health_records}}$;
- $\neg\overline{\text{Doctor}} \vee \overline{\text{Prescribe}}$;
- $\neg\overline{\text{Doctor}} \vee \overline{\text{Read_prescription}}$;
- $\neg\overline{\text{Data_Manager}} \vee \overline{\text{Read_health_records}}$;

- $\neg \overline{\text{Data_Manager}} \vee \overline{\text{Send_Data}}$;
- ... (we proceed in the same way for the remaining roles);

3. $(\neg \bar{p} \vee \bigvee \{\bar{r} : r \in R_p\})$ for all $p \in P$. In our example:

- $\neg \overline{\text{Read_id}} \vee \overline{\text{Doctor}}$;
- $\neg \overline{\text{Read_health_records}} \vee \overline{\text{Doctor}} \vee \overline{\text{Data_Manager}} \vee \overline{\text{Patient}}$;
- $\neg \overline{\text{Prescribe}} \vee \overline{\text{Doctor}}$;
- ... (we proceed in the same way for the remaining permissions).

It is easy to see that the number of clauses above is in $O(|R||P|)$ and the number of propositional variables in $O(|R| + |P|)$.

MER Constraints

As already explained in Section 4.4, the SS-HMER, the MS-DMER, and the MS-HMER can be represented as SS-DMER constraints. The first step is then to reduce them to the equivalent SS-DMER.

Let $RP = (U, R, P, UA, PA, \succeq, C)$ be an RBAC policy, $\sigma = (S, \alpha, \pi)$ be a valid state and $s \in S$. Consider the set of SS-DMER(rs, t) constraints $C[s, \sigma]$ obtained from C by replacing constraints of the form:

1. MS-DMER($rs, t \in C$ with SS-DMER($rs \setminus ur, t - |rs \cap ur|$), where
 $ur = \bigcup_{s' \in S_{user(s)} \setminus \{s\}} \alpha(s')$,
2. SS-HMER($rs, t \in C$ with SS-DMER($rs \setminus \pi(s), t - |rs \cap \pi(s)|$), and
3. MS-HMER($rs, t \in C$ with SS-DMER($rs \setminus ur, t - |rs \cap ur|$), where
 $ur = \bigcup_{s' \in S_{user(s)} \setminus \{s\}} \pi(s')$.

For all SS-DMER($rs, t \in C$ (the original ones and the ones obtained by the reduction), a CNF of the following formula is in \mathcal{C}_{RP} :

$$\sum_{r \in rs} \bar{r} \leq t - 1 \quad (5.1)$$

As shown in (79), inequalities of the form $\sum_{x \in X} x \leq t$ can be succinctly encoded into CNF with $7|X|$ clauses and $2|X|$ additional propositional variables. Thus, constraints of the

form (5.1) can be encoded with a number of variables and clauses in $O(|R|)$. The clauses are included in \mathcal{C}_{RP} as hard clauses.

In our example, the constraint SS-DMER($\{Doctor, Data_Manager\}$, 2) is encoded with 14 clauses and 4 additional propositional variables.

The User Authorization Query

A UAQ problem for the query $q = (s, P_{lb}, P_{ub}, o_p, o_r, pri)$ can be reduced to a Weighted MAX-SAT problem by adding the following hard clauses to \mathcal{C}_{RP} :

- a unit clause \bar{p} for each $p \in P_{lb}$. In our example:

- $\overline{Read_health_record};$
- $\overline{Read_prescription};$

- a unit clause $\neg\bar{p}$ for each $p \in P \setminus P_{ub}$. In our example:

- $\neg\overline{Check_process}.$

Permissions and roles optimization

The clauses in \mathcal{S} depend on the type of optimization, the objectives, and the priority. In particular, in case of:

- *permission-based optimization*, i.e. $o_p \in \{min, max\}$ and $o_r = any$: \mathcal{S} comprises

- a unit clause $\neg\bar{p}$ with weight set to 1 if $o_p = min$. In our example:

- * $\neg\overline{Read_id};$
- * $\neg\overline{Prescribe};$
- * $\neg\overline{Send_data};$
- * $\neg\overline{Manage_schedule};$

- a unit clause \bar{p} with weight set to 1 if $o_p = max$. In our example:

- * $\overline{Read_id};$
- * $\overline{Prescribe};$
- * $\overline{Send_data};$

* $\overline{\text{Manage_schedule}}$;

for each $p \in P_{ub} \setminus P_{lb}$;

- *role-based optimization*, i.e. $o_r \in \{\min, \max\}$ and $o_p = \text{any}$: \mathcal{S} comprises
 - a unit clause $\neg \bar{r}$ with weight set to 1 if $o_r = \min$ or
 - a unit clause \bar{r} with weight set to 1 if $o_r = \max$

for each $r \in R_u$;

- *joint optimization with priority to permissions*, i.e. $o_p \in \{\min, \max\}$, $o_r \in \{\min, \max\}$ and $pri = p$: \mathcal{S} comprises
 - a unit clause $\neg \bar{p}$ with weight set to $|R_u| + 1$ if $o_p = \min$ or
 - a unit clause \bar{p} with weight set to $|R_u| + 1$ if $o_p = \max$

for each $p \in P_{ub} \setminus P_{lb}$ and

- a unit clause $\neg \bar{r}$ with weight set to 1 if $o_r = \min$ or
- a unit clause \bar{r} with weight set to 1 if $o_r = \max$

for each $r \in R_u$;

- *joint optimization with priority to roles*, i.e. $o_p \in \{\min, \max\}$, $o_r \in \{\min, \max\}$ and $pri = r$: \mathcal{S} comprises
 - a unit clause $\neg \bar{r}$ with weight set to $|P_{ub} \setminus P_{lb}| + 1$ if $o_r = \min$ or
 - a unit clause \bar{r} with weight set to $|P_{ub} \setminus P_{lb}| + 1$ if $o_r = \max$

for each $r \in R_u$ and

- a unit clause $\neg \bar{p}$ with weight set to 1 if $o_p = \min$ or
- a unit clause \bar{p} with weight set to 1 if $o_p = \max$

for each $p \in P_{ub} \setminus P_{lb}$;

- *no optimization*, i.e. $o_p = o_r = \text{any}$: no soft clauses are included in \mathcal{S} , i.e. $\mathcal{S} = \emptyset$.

As explained above, in case of joint optimization there are $|P_{ub} \setminus P_{lb}|$ clauses for extra permissions and $|R_u|$ clauses for extra roles. The weights given to extra permissions (W_p) and the extra roles (W_r) are chosen to ensure the proper prioritization:

- when $pri = p$, we must ensure that the solver tries to satisfy any clause on any permission in $P_{ub} \setminus P_{lb}$ before satisfying any clause on extra roles. As a consequence, we must ensure that W_p is higher than the sum of the weights given to the $|R_u|$ clauses on extra roles, which is $W_r = 1$ for the sake of simplicity². For this reason, we set:

$$W_p = (|R_u| \times W_r) + 1 = |R_u| + 1$$

- when $pri = r$, we must ensure that the solver tries to satisfy any clause on any extra role before satisfying any clause on extra permissions. As a consequence, we must ensure that W_r is higher than the sum of the weights given to the $|P_{ub} \setminus P_{lb}|$ clauses on extra permissions, which is $W_p = 1$ ³. For this reason, we set:

$$W_r = (|P_{ub} \setminus P_{lb}| \times W_p) + 1 = |P_{ub} \setminus P_{lb}| + 1$$

For the sake of completeness, Tables 5.2 and 5.3 report the clauses on extra permissions and roles in case of joint optimization, respectively with $pri = p$ and $pri = r$. In the former table, the clauses highlighted in gray have weight $|R_u| + 1 = 3 + 1 = 4$, while in the latter table they have weight $|P_{ub} \setminus P_{lb}| + 1 = 6 - 2 + 1 = 5$. In both tables, the remaining clauses have weight 1.

Finally, in Table 5.4, we present the clauses on extra roles in case of role-based optimization. These clauses have weight 1.

As already stated in Section 4, in this work we considered only the permission-based optimization and the joint optimization with $pri = p$, because they are the most relevant. However, the methodology presented can be extended and applied also for the other types of optimization.

²In fact, since the optimization of roles is the secondary objective, it is enough that the aforementioned condition between W_p and W_r is respected.

³Similarly to the previous case.

Table 5.2 Conditions on extra permissions and roles in case of joint optimization with $pri = p$

$o_p = min$	$o_p = min$	$o_p = max$	$o_p = max$
$o_r = min$	$o_r = max$	$o_r = min$	$o_r = max$
$pri = p$	$pri = p$	$pri = p$	$pri = p$
$\neg \overline{Read_id}$	$\neg \overline{Read_id}$	$Read_id$	$Read_id$
$\neg \overline{Prescribe}$	$\neg \overline{Prescribe}$	$Prescribe$	$Prescribe$
$\neg \overline{Send_data}$	$\neg \overline{Send_data}$	$Send_data$	$Send_data$
$\neg \overline{Doctor}$	\overline{Doctor}	$\neg \overline{Doctor}$	\overline{Doctor}
$\neg \overline{Data_manager}$	$\overline{Data_manager}$	$\neg \overline{Data_manager}$	$\overline{Data_manager}$
$\neg \overline{Head_Physician}$	$\overline{Head_Physician}$	$\neg \overline{Head_Physician}$	$\overline{Head_Physician}$

Table 5.3 Conditions on extra permissions and roles in case of joint optimization with $pri = r$

$o_p = min$	$o_p = min$	$o_p = max$	$o_p = max$
$o_r = min$	$o_r = max$	$o_r = min$	$o_r = max$
$pri = r$	$pri = r$	$pri = r$	$pri = r$
$\neg \overline{Read_id}$	$\neg \overline{Read_id}$	$Read_id$	$Read_id$
$\neg \overline{Prescribe}$	$\neg \overline{Prescribe}$	$Prescribe$	$Prescribe$
$\neg \overline{Send_data}$	$\neg \overline{Send_data}$	$Send_data$	$Send_data$
$\neg \overline{Doctor}$	\overline{Doctor}	$\neg \overline{Doctor}$	\overline{Doctor}
$\neg \overline{Data_manager}$	$\overline{Data_manager}$	$\neg \overline{Data_manager}$	$\overline{Data_manager}$
$\neg \overline{Head_Physician}$	$\overline{Head_Physician}$	$\neg \overline{Head_Physician}$	$\overline{Head_Physician}$

Table 5.4 Conditions on extra roles in case role-based optimization

$o_p = \text{any}$	$o_p = \text{any}$
$o_r = \text{min}$	$o_r = \text{max}$
$pri = r$	$pri = r$
<hr/>	<hr/>
$\neg \overline{\text{Doctor}}$	$\overline{\text{Doctor}}$
$\neg \overline{\text{Data_manager}}$	$\overline{\text{Data_manager}}$
$\neg \overline{\text{Head_Physician}}$	$\overline{\text{Head_Physician}}$
<hr/>	<hr/>

Chapter 6

Benchmarks

Designing benchmarks suitable for the systematic assessment of UAQ solvers is not easy. A common approach (13; 53; 64; 90) is to focus on families of problems that are parametric on aspects of the problem that may contribute to its complexity. All other aspects are either set to a predefined, constant value or are randomly chosen in a given interval or according to some criterion. By running a solver against the instances corresponding to increasing values of the parameter, it is thus possible to obtain an estimation of how the solver scales along the dimension represented by the parameter. Unfortunately, the adequacy of the benchmarks proposed in the literature is seldom discussed. The lack of adequate benchmarks makes it difficult to assess the efficiency of the different techniques proposed to tackle the UAQ problem. Even more so, it makes it difficult to compare them.

The sheer number of elements that contribute to the definition of the UAQ problem complicates the selection of the parameters. The elements characterizing the RBAC policy include the number of roles $|R|$, the number of permissions $|P|$, the number of DMER constraints $|C|$ as well as their specific features, namely the maximum number of roles participating in each constraint $\hat{r}s$ and the maximum bound \hat{t} . One may even consider features of the PA relation, such as the maximum number of roles that contain any given permission, referred as $\widehat{R}_P = \max_{p \in P} |R_p|$ in Section 5. The components of the query also contribute to the complexity of the problem. These include the security objective(s) (any, min, max), the number of requested permissions that *must* be granted, i.e. $|P_{lb}|$, and the number of requested permissions that *can* be granted, i.e. $|P_{ub}|$.

6.1 A new approach to UAQ benchmark evaluation and generation

As already stated in Section 2.3, the UAQ problem has been demonstrated to be NP-hard. In Section 5.1 we introduced the best algorithms known to tackle it. The asymptotic time complexity of these algorithms can be used to discriminate classes of UAQ problems that can be hard to solve from the ones that should be easy to solve and they can be therefore used to validate and even guide the design of benchmarks for the UAQ problem. Once validated, the benchmarks can be used to assess and compare the performance of the solvers.

The algorithms introduced in Section 5.1 play a key role in the proposed methodology, since they provide upper bounds on the asymptotic growth rate for UAQ solvers which, to the best of our knowledge, are the best upper bounds currently available in the literature. Yet, it must be noted that the proposed methodology will yield different, improved results as soon as new complexity results will become available. Thus the benchmarks proposed and used in this thesis could be improved consequently.

Benchmark Evaluation. Let \mathcal{Q} be an infinite set of problems and $\mathcal{Q}[n] \subseteq \mathcal{Q}$ for all $n \in \mathbb{N}$ a parametric family of finite sets of formulae in \mathcal{Q} . Let S be the (infinite) set of all solvers for \mathcal{Q} . By $t_s(q)$ we denote the time spent by solver $s \in S$ to solve problem $q \in \mathcal{Q}$. Let $\mathcal{Q}' \subset \mathcal{Q}$ be a *finite* subset of \mathcal{Q} .

Methodology.

- If \mathcal{Q} is known to be *NP-hard*:
 - If $t_s(\mathcal{Q}[n])$ increases exponentially as n increases for all $s \in S$, then $\mathcal{Q}[n]$ with $n \in \mathbb{N}$ is adequate to represent the complexity of \mathcal{Q} ;
 - If there exists $s \in S$ such that $t_s(\mathcal{Q}[n])$ grows in polynomial time, then $\mathcal{Q}[n]$ with $n \in \mathbb{N}$ is a tractable subclass \mathcal{Q}' of \mathcal{Q} and, under the assumption that $P \neq NP$, we can conclude that it is inadequate to represent the complexity of \mathcal{Q} .
- If \mathcal{Q} is known to be in P , then by definition we know that there exists a solver $s \in S$ such that $t_s(\mathcal{Q}[n])$ increases in polynomial time. If $s \in S$ is such that $t_s(\mathcal{Q}[n])$ grows exponentially, we can then conclude that s is inefficient against the problems in \mathcal{Q} .

As it is, the above methodology cannot be applied in practice as it implies the execution of an infinite number of solvers (i.e., all $s \in S$) against an infinite number of problems (i.e.,

all $q \in Q[n]$, for all $n \in \mathbb{N}$). Yet, from it we can derive an approximate, practical version by restricting the application to a finite set of solvers $S_0 \subset S$ against a finite family of finite sets $Q_0[n] \in Q$ with $n \in [0..n_0]$ and $n_0 \in \mathbb{N}$.

Practical Methodology

- If Q is known to be *NP-hard*, then
 - if $t_s(Q_0[n])$ grows exponentially in $[0..n_0]$ for all $s \in S_0$, then we conclude that the family of problems $Q_0[n]$ with $n \in [0..n_0]$ is *empirically adequate* to represent the complexity of Q ; notice that this does not imply that $Q_0[n]$ with $n \in [0..n_0]$ adequately represents the complexity of Q as some solver in S could exhibit a sub-exponential behaviour for $n > n_0$;
 - if there exists $s \in S_0$ such that $t_s(Q_0[n])$ increases in polynomial time in $[0..n_0]$, then we conclude that $Q_0[n]$ with $n \in [0..n_0]$ is *empirically inadequate* to represent the complexity of Q ; notice that this does not necessarily imply that $Q_0[n]$ with $n \in [0..n_0]$ is inadequate to represent Q as all solvers in S could exhibit an exponential behaviours for $n > n_0$ (or $P = NP$).
- If Q is known to be in P :
 - If $t_s(Q_0[n])$ increases polynomially in $[0..n_0]$ for all $s \in S_0$, then we conclude that $Q_0[n]$ with $n \in [0..n_0]$ is empirically adequate to represent the complexity of Q ; notice that this does not imply that $Q_0[n]$ with $n \in [0..n_0]$ *adequately represents* the complexity of Q as all solvers in S could exhibit an exponential behaviour for $n > n_0$;
 - For all solvers $s \in S_0$ such that $t_s(Q_0[n])$ grows exponentially in $[0..n_0]$, we conclude that they are *empirically inefficient* over the problems in Q ; notice that this does not necessarily imply that these solvers are inefficient, as they could have a polynomial behaviour for $n > n_0$.

For the sake of readability, in the future sections we will use the expression "the benchmark is empirically adequate (/inadequate)" meaning that the benchmark adequately represents (/does not represent) the complexity of the UAQ problem by assuming that:

- the solvers used in the experiments exhibit the same behavior when the parameter is outside the interval defined for the benchmark;

- the other solvers exhibit the same behavior as the ones executed for the experiments.

As the same way, we will use the expression "the solver is empirically inefficient over the class of problems" meaning that the solver is inefficient over the class of UAQ problem assuming that the solver does not exhibit a polynomial behavior outside the interval defined for the benchmark.

The applicable methodology is summarized in Table 6.1. For example, suppose we have a benchmark parametric in $|R|$ and that the best algorithm known to solve it is exponential in $|R|$. We then expect that any solver running over it exhibits an exponential growth with the increase in $|R|$. If this is the case, we say that the benchmark is empirically adequate; otherwise we say that it is empirically inadequate. Dually, if the best algorithm known to solve the benchmark is polynomial in $|R|$, we expect that any solver solves the benchmark in polynomial time. If this is the case, we say that the benchmark is empirically adequate; otherwise, we say that the solver is empirically inefficient against the class of UAQ problems.

Table 6.1 Methodology to evaluate benchmarks

Expected result	Experimental result	Conclusions
Exponential	Polynomial	The benchmark is empirically inadequate
	Exponential	The benchmark is empirically adequate
Polynomial	Polynomial	The benchmark is empirically adequate
	Exponential	The solver is empirically inefficient against this class of UAQ problems

Benchmark Generation. A complete set of benchmarks should contain both hard and easy instances: the first ones are useful to stress-test the solvers, while the second ones can be used to check the efficiency of the solvers over simple problems.

If we want to generate a benchmark parametric in a certain dimension highlighting the behavior of one of the algorithms introduced in Section 5.1, the algorithms themselves can help us to size the other problem dimensions adequately.

For example, we may want to generate a benchmark whose complexity grows exponentially with $|P_{lb}|$ when $o_p = \min$, behaving as Algorithm 3. To do so, we should choose a value for $|R|$ so that $\widehat{R}_P^{|P_{lb}|} < 2^{|R|}$, otherwise, any efficient solver would behave like Algorithm 1, whose complexity does not depend on $|P_{lb}|$.

On the contrary, if we want to generate an easy benchmark parametric in $|P_{lb}|$, it is necessary to ensure that $\widehat{R}_P^{|P_{lb}|} > 2^{|R|}$. In fact, in this case, any efficient solver would behave like Algorithm 1.

The methodology presented above to generate benchmarks of problems with $o_p = \min$ and $o_p = \max$ is summarized respectively in Tables 6.2 and 6.3.

Note that, when $o_p = \min$, Algorithm 3 should always perform better than Algorithm 2, since $2^{\widehat{R}_P^{|P_{ub}|}} > \widehat{R}_P^{|P_{lb}|}$. To see this, we observe that from $2^{\widehat{R}_P^{|P_{ub}|}} > \widehat{R}_P^{|P_{lb}|}$ implies $\widehat{R}_P^{|P_{ub}|} > \log_2 \widehat{R}_P^{|P_{lb}|}$ which in turn implies $|P_{ub}| > |P_{lb}| \frac{\log_2 \widehat{R}_P}{\widehat{R}_P}$. This is always true since $P_{lb} \subseteq P_{ub}$ (and consequently $|P_{lb}| \leq |P_{ub}|$) and $\frac{\log_2 x}{x} < 1$. Moreover, when $o_p = \max$, Algorithm 1 should always perform better than Algorithm 2, because $2^{\widehat{R}_P^{|P_{ub}|}} < 2^{|R|}$ is not acceptable. In fact, $2^{\widehat{R}_P^{|P_{ub}|}} < 2^{|R|}$ would imply $\widehat{R}_P^{|P_{ub}|} < |R|$. Since we consider $|P_{ub}| = |P|$, we would have $\widehat{R}_P^{|P|} < |R|$. This would mean having far more roles than permissions. It is easy to see that in this case we would have many roles with no permission assigned, which is absurd. This means that any reasonably performing solver should never exhibit exponential growth over benchmarks parametric in \widehat{R}_P or $|P_{ub}|$. Since Algorithm 2 should never be the most performing algorithm for both $o_p = \min$ and $o_p = \max$, no benchmark should stimulate an exponential growth with the increase in \widehat{R}_P or $|P_{ub}|$ behaving like Algorithm 2 ($O(2^{\widehat{R}_P^{|P_{ub}|}})$). For this reason, Algorithm 2 does not contribute to our methodology and then does not appear in Tables 6.2 and 6.3.

The methodology described above is used in Section 6.2 to evaluate the benchmarks from (64) and in Section 6.3 to guide the design of new benchmarks. The benchmarks evaluation is confirmed by the experimental results presented in Section 8.1.

Table 6.2 Methodology to generate benchmarks with $o_p = \min$

Parameter	Complexity	Algorithm	Conditions
$ P_{lb} $	Hard	3	$ R $ and \widehat{R}_P must be set so that $\widehat{R}_P^{ P_{lb} } < 2^{ R }$
	Easy	1	$ R $ and \widehat{R}_P must be set so that $\widehat{R}_P^{ P_{lb} } > 2^{ R }$
$ R $	Hard	1	\widehat{R}_P and $ P_{lb} $ must be set so that $2^{ R } < \widehat{R}_P^{ P_{lb} }$
	Easy	3	\widehat{R}_P and $ P_{lb} $ must be set so that $2^{ R } > \widehat{R}_P^{ P_{lb} }$
\widehat{R}_P	Hard	3	$ R $ and $ P_{lb} $ must be set so that $\widehat{R}_P^{ P_{lb} } < 2^{ R }$
	Easy	1	$ R $ and $ P_{lb} $ must be set so that $\widehat{R}_P^{ P_{lb} } > 2^{ R }$
$ C $	Hard	Not applicable	Those parameters do not contribute to the UAQ problem complexity for $o_p = \min$, because they do not appear in the complexity results for $o_p = \min$
	Easy	Always	
\widehat{rs}	Hard	Not applicable	
	Easy	Always	
\widehat{t}	Hard	Not applicable	
	Easy	Always	

Table 6.3 Methodology to generate benchmarks with $o_p = max$

Parameter	Complexity	Algorithm	Conditions
$ P_{lb} $	Hard	Not applicable	$ P_{lb} $ does not contribute to the UAQ problem complexity for $o_p = max$, because it does not appear in the complexity results for $o_p = max$
	Easy	Always	
$ R $	Hard	1	$ C $, \hat{rs} and \hat{t} must be set so that $2^{ R } < \hat{rs}^{ C \hat{t}}$
	Easy	4	$ C $, \hat{rs} and \hat{t} must be set so that $2^{ R } > \hat{rs}^{ C \hat{t}}$
\widehat{R}_P	Hard	Not applicable	\widehat{R}_P does not contribute to the UAQ problem complexity for $o_p = max$, because Algorithms 1 and 4 should always perform better than Algorithm 2.
	Easy	Always	
$ P_{ub} $	Hard	Not applicable	$ P_{ub} $ does not contribute to the UAQ problem complexity for $o_p = max$, because Algorithms 1 and 4 should always perform better than Algorithm 2.
	Easy	Always	
$ C $	Hard	4	$ R $, \hat{rs} and \hat{t} must be set so that $\hat{rs}^{ C \hat{t}} < 2^{ R }$
	Easy	1	$ R $, \hat{rs} and \hat{t} must be set so that $\hat{rs}^{ C \hat{t}} > 2^{ R }$
\hat{rs}	Hard	4	$ R $, $ C $ and \hat{t} must be set so that $\hat{rs}^{ C \hat{t}} < 2^{ R }$
	Easy	1	$ R $, $ C $ and \hat{t} must be set so that $\hat{rs}^{ C \hat{t}} > 2^{ R }$
\hat{t}	Hard	4	$ R $, $ C $ and \hat{rs} must be set so that $\hat{rs}^{ C \hat{t}} < 2^{ R }$
	Easy	1	$ R $, $ C $ and \hat{rs} must be set so that $\hat{rs}^{ C \hat{t}} > 2^{ R }$

6.2 Benchmarks from (64)

In previous work (see, e.g., (13; 53; 64; 90)), various benchmark problems parametric in any of the aforementioned dimensions have been put forward. To the best of our knowledge, the most extensive collection of parametric benchmarks so far is presented in (64) and summarized in Table 6.4.

Table 6.4 Parametric Benchmarks from (64)

Name	$ R $	$ P $	\widehat{R}_P	$ C $	\widehat{rs}	\widehat{t}	$ P_{lb} $	$ P_{ub} $
roles	25..200	500	3	10	10	3	7	20
d	100	500	3	10..100	10	3	7	23
rolesPerConstr	300	1000	3	20	10..100	3	5	30
t	100	500	3	20	25	2..12	6	10
plb	100	500	3	10	10	3	1..11	$20 - P_{lb} $

To illustrate, consider the benchmark problems named “roles” in the table. They are parametric in $|R|$ (with $|R|$ ranging from 25 to 200), have 500 permissions ($|P|$) with every permission being assigned to exactly 3 roles (\widehat{R}_P), and 10 MER constraints ($|C|$); each MER constraint involves 10 roles (\widehat{rs}) with bound 3 (\widehat{t}). The cardinality of P_{lb} and P_{ub} are set to 7 and 20 respectively. Only the optimization objective min is considered.

Table 6.5 compares the complexities of Algorithms 1 and 3 for all the benchmarks proposed in (64). Algorithm 4 is not taken into account, because it solves only problems with $o_p = \max$.

Table 6.5 Comparing algorithm performances

Name	A1: $2^{ R }$	A3: $\widehat{R}_P^{ P_{lb} }$
roles	$3.4 \times 10^7 \dots 1.6 \times 10^{60}$	2187
d	1.5×10^{30}	2187
rolesPerConstr	2×10^{90}	243
t	1.3×10^{30}	729
plb	1.3×10^{30}	$3..1.8 \times 10^5$

From Table 6.5, we can observe that Algorithm 3 is likely to be the most efficient for all the benchmarks. Besides, in all the benchmarks except plb the value of $|P_{lb}|$ is fixed.

Algorithm 3 is therefore insensitive to the value of the respective parameter ($|R|$ for roles, $|C|$ for d, $\hat{r}s$ for rolesPerConstr, and \hat{t} for t) and so it should be reasonably efficient when applied to these problems. We can expect that all the benchmarks but plb can be used to check whether UAQ solvers are as effective as Algorithm 3 as the value of the respective parameter increases. On the contrary, benchmark plb is parametric in $|P_{lb}|$, therefore we expect that the solving time of Algorithm 3 (and of any other solver) increases exponentially as $|P_{lb}|$ increases.

While the benchmarks in (64) are a first attempt to provide a comprehensive evaluation along a number of significant dimensions, they still suffer from the following shortcomings:

1. only the optimization objective min is considered, and they are therefore not suitable for evaluating the performance of the solvers when different optimization objectives, most notably max, are considered;
2. it is not always clear if these benchmarks are adequate.

6.3 The new benchmarks

Driven by the methodology introduced in Section 6.1, we propose two new sets of parametric UAQ problems, one with $o_p = \min$ and one with $o_p = \max$. The benchmarks with $o_p = \min$ are summarized in Table 6.6 and described as follows:

Table 6.6 Benchmark specifications for $o_p = \min$

Name	$ R $	$ P_{ub} $	\widehat{R}_P	$ C $	$\hat{r}s$	\hat{t}	$ P_{lb} $
<i>Plb_bigR</i>	200	400	5	0	-	-	5..50
<i>Plb_smallR</i>	10	400	5	0	-	-	5..50
<i>R_bigPlb</i>	10..100	400	5	0	-	-	100
<i>R_smallPlb</i>	10..100	400	5	0	-	-	2
<i>RPhat_bigPb</i>	200	400	2..12	0	-	-	10
<i>RPhat_medPb</i>	200	400	2..12	0	-	-	4
<i>RPhat_smallPb</i>	200	400	2..12	0	-	-	1
<i>Pub</i>	200	100..1000	5	50	8	3	10
<i>C</i>	200	400	5	10..100	8	3	10
<i>rshat</i>	100	400	5	10	5..50	3	10
<i>that</i>	1000	1000	1	50	20	2..8	10

- Plb_bigR and Plb_smallR are both parametric in $|P_{lb}|$. Plb_bigR can be used to stress-test solvers for increasing values of $|P_{lb}|$: for large values of $|R|$ (here set to 200), the best known algorithm (i.e. Algorithm 3) is exponential in $|P_{lb}|$ and thus we expect any solver to exhibit the same behavior. Plb_smallR can instead be used to check the effectiveness of solvers: Algorithm 1 is in $O(2^{|R|})$ and thus we know that the problem can be solved efficiently for sufficiently small values of $|R|$ (here set to 10).
- R_bigPlb and $R_smallPlb$ are both parametric in $|R|$ and are dual to Plb_bigR and Plb_smallR respectively. R_bigPlb can be used to stress-test solvers for increasing values of $|R|$: for large values of $|P_{lb}|$ (here set to 100), the best known algorithm (i.e. Algorithm 1) is exponential in $|R|$ and thus we expect any solver to exhibit the same behavior. $R_smallPlb$ can be used to check the efficiency of solvers: Algorithm 3 is in $O(\widehat{R}_P^{|P_{lb}|})$ and thus we know that the problem can be solved efficiently for sufficiently small values of $|P_{lb}|$ (here set to 2).
- $RPhat_bigPlb$, $RPhat_medPlb$, $RPhat_smallPlb$ are parametric in \widehat{R}_P and can be used to check the effectiveness of solvers: Algorithm 3 is in $O(\widehat{R}_P^{|P_{lb}|})$ and thus we know that the problem can be solved efficiently for sufficiently small values of $|P_{lb}|$ (here set to 10, 4, and 1 respectively). Note that $|P_{lb}|$ is the degree of the polynomial and therefore the time spent by the solver may differ significantly (for the values of $|P_{lb}|$ considered) as \widehat{R}_P increases.
- Pub is parametric in $|P_{ub}|$. As already noted in Section 6.1, Algorithm 2 should always perform worst than Algorithm 3. As a consequence, $|P_{ub}|$ should not contribute to the UAQ problem complexity. Therefore this benchmark can be used to check the effectiveness of solvers.
- C , $rshat$, and $that$, are parametric in $|C|$, \widehat{rs} and \widehat{t} respectively. Since for $o_p = \min$ these parameters do not contribute to the asymptotic complexity of any algorithm presented in Section 5.1, these benchmarks can be used to check the effectiveness of solvers.

Notice that we do not include benchmarks parametric in the “size” of the role hierarchy since, as already pointed out in Section 5, it does not contribute to the complexity of the UAQ problem.

The benchmarks with $obj = \max$ are summarized in Table 6.7 and described below:

Table 6.7 Benchmark specifications for $o_p = \max$

Name	$ R $	$ P_{ub} $	\widehat{R}_P	$ C $	\widehat{rs}	\widehat{t}	$ P_{lb} $
R_bigCt	10..100	400	5	50	8	3	10
$R_smallCt$	10..100	400	5	5	3	2	10
Pub	200	100..1000	5	50	8	3	10
$RPhat$	200	400	20..60	50	25	4	4
C_bigR	200	400	5	10..100	8	3	10
C_smallR	10	400	5	10..100	8	3	10
$that_bigR$	1000	1000	1	50	20	2..12	10
$that_smallR$	20	400	5	10	12	2..12	10
$rshat_bigCt$	200	400	5	10	5..50	3	10
$rshat_medCt$	200	400	5	3	5..50	3	10
$rshat_smallCt$	200	400	5	1	5..50	3	10
Plb	200	400	5	20	5	2	5..50

- R_bigCt and $R_smallCt$ are parametric in $|R|$. R_bigCt is designed to stress test solvers for increasing values of $|R|$: when $|C|$ and \widehat{t} are such that $|C|\widehat{t}$ is sufficiently large (here set to 150), the best algorithm (Algorithm 1) is exponential in $|R|$ and thus we expect any solver to exhibit the same behavior. $R_smallCt$ is instead designed to check the effectiveness of solvers: when $|C|$ and \widehat{t} are such that $|C|\widehat{t}$ is sufficiently small (here set to 10), Algorithm 4 can efficiently solve the problems independently from the size of $|R|$ and any efficient solver should do the same.
- Pub is parametric in $|P_{ub}|$. Since Algorithm 1 should perform better than Algorithm 2, Pub can instead be used to check the efficiency of solvers.
- $RPhat$ is parametric in \widehat{R}_P and is analogous to the previous case. We then expect that $RPhat$ can be solved efficiently.
- C_bigR and C_smallR are parametric in $|C|$. C_bigR can be used to stress-test solvers for increasing values of $|C|$: for large values of $|R|$ and $|P_{ub}|$ Algorithm 4 is to be preferred to Algorithm 1. Since Algorithm 4 is exponential in $|C|$, we expect solvers to exhibit the same behavior. C_smallR can be used to check the efficiency of solvers: Algorithm 1 is in $O(2^{|R|})$ and thus the problem can be solved efficiently for sufficiently small values of $|R|$.

- *that_bigRPub* and *that_smallR*, parametric in \hat{t} , are analogous to the previous case. We then expect that solvers exhibit exponential growth over *that_bigRPub*, while *that_smallR* can be solved efficiently.
- *rshat_smallCt*, *rshat_medCt* and *rshat_bigCt* are parametric in \hat{rs} . *rshat_smallCt* can be used to check the effectiveness of solvers. Algorithm 4 is in $O(\hat{rs}^{|C|\hat{t}})$ and thus the problem can be solved efficiently for sufficiently small values of $|C|\hat{t}$. *rshat_medCt* and *rshat_bigCt* can be used to see how the values of $|C|\hat{t}$ affect the complexity of the problem.
- *Plb* is parametric in $|P_{lb}|$. Since for $o_p = \max$ this parameter does not contribute to the asymptotic complexity of any algorithm presented in Section 5.1, this benchmark can be used to check the effectiveness of solvers.

The benchmarks designed in Table 6.6 and in Table 6.7 have been generated through UAQ-Solve, a tool which will be introduced in Section 7.1. Both UAQ-Solve and the benchmarks are available online¹.

Let us consider *Plb* (the last benchmark in Table 6.7) as example. The benchmark is parametric in $|P_{lb}|$, which ranges from 5 to 50, with a step of 5 (not precised in the table). For any value of $|P_{lb}| x$, we generate 10 instances. For any instance, we create an RBAC policy considering $|R| = 200$ roles and $|P| = |P_{ub}| = 400$ permissions. In particular, any permission is assigned to exactly $\widehat{R}_P = 5$ roles.

Then, we generate $|C| = 20$ SS-DMER(rs, \hat{t}) constraints with $t = \hat{t} = 2$: for each constraint we randomly select $\hat{rs} = |rs| = 5$ among the $|R|$ roles.

We then create a query consisting of $|P_{lb}| = x$ permissions and $|P_{ub}| = |P|$ permissions. In fact, as already mentioned in Section 5, all the permissions in $P \setminus P_{ub}$ must not be granted; therefore all the roles that activate these permissions cannot be included in a solution. These roles and the permissions in $P \setminus P_{ub}$ can thus be safely removed from the policy (in polynomial time). By doing that, we simply exclude UAQ problems that are de facto equivalent to UAQ problems that meet our assumptions.

In Sections 8.2.1 and 8.3.1 we will evaluate our benchmarks and we will prove that they are empirically adequate. The introduction of these suite of benchmarks is an important contribution. In fact, the lack of adequate benchmarks made it difficult to assess the efficiency of the different techniques used to tackle the UAQ problem. Even more so, it made it difficult to compare their performance. By providing a complete and empirically adequate suite of benchmarks, we are de facto offering the possibility to both

¹https://github.com/GioGazza/uaq_prolem.git

- assess the performance of a single solver, and
- compare the performance of the different solutions.

Chapter 7

A MaxSAT-based solver

7.1 UAQ-Solve

UAQ-Solve is a command-line tool implemented in Perl and C. It is freely available online¹. We implemented UAQ-Solve to fulfill two functionalities: generating new benchmarks and solving benchmarks. We used UAQ-Solve to generate a set of hard UAQ problem instances that have been used to stress-test solvers at the MaxSAT Evaluation 2018 (10).

7.1.1 UAQ-Solve as benchmarks generator

Following the common approach (13; 53; 64; 90), UAQ-Solve focuses on families of problems that are parametric on aspects of the UAQ problem that may contribute to its complexity; namely, each benchmark generated by UAQ-Solve is parametric in only one dimension. All the other aspects are set to a predefined, constant value. The following dimensions are provided as input to UAQ-Solve thorough a specification file:

- the number of roles $|R|$;
- the number of permissions $|P|$;
- the number of roles a permission is assigned to \widehat{R}_P ;
- the minimum number of permissions assigned to each role;
- the number of MER constraints $|C|$;
- the number of roles bounded by each MER constraint \widehat{rs} ;

¹https://github.com/GioGazza/uaq_prolem.git.

- the MER bound \hat{t} ;
- the number of permissions whose activation is requested $|P_{lb}|$;
- the permissions upper bound $|P_{ub}|$.

For each UAQ problem instance, UAQ-Solve first creates the permission assignment relations among the $|R|$ roles and the $|P|$ permissions. The relations are randomly selected, however UAQ-Solve eventually discharges the relations that violate the clause on the \widehat{R}_P , namely if the permission considered in the relation is already assigned to \widehat{R}_P roles.

UAQ-Solve then generates $|C|$ SS-DMER constraints: for all of them, it randomly selects the set rs . The value for t is the one provided in the specification file.

Finally, it generates the query, in which $|P_{lb}|$ permissions are requested and $|P| \setminus |P_{ub}|$ permissions are denied.

For the sake of completeness, we provide an example of specification file in Listing 7.1 and one of the resulting instances of UAQ problem generated by UAQ-Solve in Listing 7.2. With the specification file in Listing 7.1, UAQ-Solve generates a benchmark parametric in the number of roles $|R|$. In particular, the parameter ranges from 5 (ROLES_MIN = 5) to 15 (ROLES_MAX = 15), with step 5 (ROLES_STEP = 5). For every step, UAQ-Solve generates 3 instances (INSTANCES_MAX - INSTANCES_MIN = 3).

Listing 7.1 Specification file

```

1 -- INSTANCES_MIN=0 -- INSTANCES_MAX=3 -- SESSIONS_MAX=1
2 -- ROLES_MIN=5 -- ROLES_MAX=15 -- ROLES_STEP=5
3 -- NUM_PERMS=10 -- PERMS_PER_ROLE=1 -- ROLES_PER_PERM=2
4 -- NUM_MERS=1 -- ROLES_PER_CONSTR=2 -- MER_BOUND=2
5 -- PERMS_LB_START=2 -- PERMS_UB=9

```

Listing 7.2 is one of the instances generated with 5 roles (r1, r2, ..., r5 in row 2). Besides, the instance has 10 permissions (NUM_PERMS = 10, resulting in p1, p2, ..., p10 in row 3). As visible in rows 10 - 14, each role has assigned at least one permission (PERMS_PER_ROLE = 1), while each permission is assigned exactly to 2 roles (ROLES_PER_PERM = 2). The problem is constrained by 1 Single-Session DMER (NUM_MERS = 1), which allows the activation of only 1 role (MER_BOUND = 2) among the 2 roles (ROLES_PER_CONSTR = 2) r2 and r4, see row 18. Row 20 presents the query, in which 2 permissions (p1 and p2) are requested (PERMS_LB_START = 2) and 1 permission (p3) is denied (NUM_PERMS - PERMS_UB = 1).

Listing 7.2 Instance of UAQ problem generated by UAQ-Solve

```

1 users : alice ;
2 roles : r1 r2 r3 r4 r5 ;
3 perms : p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 ;
4 sesss : s1 ;

5
6 sof [ s1 ] : alice ;
7 --
8 ua [ alice ] : r1 r2 r3 r4 r5 ;
9 --
10 pa [ r1 ] : p10 ;
11 pa [ r2 ] : p5 p7 p9 ;
12 pa [ r3 ] : p1 p2 p3 p4 p6 p8 p9 p10 ;
13 pa [ r4 ] : p2 p3 p4 p6 p7 p8 ;
14 pa [ r5 ] : p1 p5 ;
15 --
16 --
17 --
18 mer ss d 2 r2 r4 ;
19 --
20 QUERY s1 MIN GRANT p1 p2 DENY p3 ;

```

7.1.2 UAQ-Solve as benchmarks solver

UAQ-Solve is also a solver for the UAQ problem that implements a reduction of the UAQ problem to Partial Maximum Satisfiability (PMaxSAT) problem in case of permission-based optimization or to Partial Weighted Maximum Satisfiability (WPMaxSAT) problem in case of joint optimization. The reduction enables UAQ-Solve to leverage state-of-the-art (W)PMaxSAT solvers taken off-the-shelf to solve UAQ problems. This approach takes advantage of the fast-paced advancements achieved by a lively and dedicated research community that organizes the MaxSAT Evaluation on a yearly basis since 2006². The MaxSAT Evaluation (MSE) is the primary competition like event focusing on the evaluation of MaxSAT solvers. Its declared goals are:

- the assessment of state-of-the-art MaxSAT solvers;
- the promotion of MaxSAT as a suitable solution for solving instances of a wide range of NP-hard optimization problems;

²<https://maxsat-evaluations.github.io/>

- the collection and distribution of valuable and heterogeneous benchmarks for further scientific evaluations.

To these aims, every year a call for solvers and a call for benchmarks encoding instances of interesting NP-hard optimization problems are opened. Specifically, the proposed benchmarks are used to evaluate the performance of the different solvers. Solvers can be submitted to different tracks: one of the most interesting for our scopes is the incomplete track, which is dedicated to incomplete solvers. In contrast to the complete solvers, which execute until they find an optimum solution, the incomplete solvers execute at most until a pre-set timeout: if they find an optimum solution before the timeout, they return it; otherwise, they return the best solution found.

The MaxSAT Evaluation mandates the participants to follow a set of rules, which are published in the MaxSAT Evaluation web page³: in particular, it establishes the input and output formats that all the participant solver must conform to. The PMaxSAT problem instance must be read from the file given as a parameter to the solver. The format chosen for encoding the PMaxSAT problem is the Weighted Conjunctive Normal Form (WCNF), which is characterized by the following parameters line (first row of the file):

p wcnf *nbvar* *nbclauses* *top*

where *nbvar* is the number of variables, *nbclauses* is the number of clauses and *top* is the weight given to the hard clauses. In particular, *top* must be greater than the sum of the weights given to the soft clauses. After the parameters line, one clause per row follows. Each clause is a disjunction of variables and is delimited by the weight of the clause (at the beginning of the row) and a 0 (in the end of the row). Each variable is labelled with an integer number.

Regarding the output, it must consist of the following elements:

- Comments: optional lines starting with "c " and containing extra information;
- Solution Status: mandatory line starting with "s " followed by the answer of the solver, which must be one among the following: "OPTIMUM FOUND" when the optimum solution is found, "UNSATISFIABLE" when there is no truth assignment satisfying the hard clauses, and "UNKNOWN" otherwise. The incomplete solvers typically return "UNKNOWN" unless they can verify that the best solution found corresponds to the optimum one;

³<https://maxsat-evaluations.github.io/2019/rules.html>

- Solution Cost: mandatory line starting with "o" followed by an integer representing the solution cost, namely the sum of the weights of the clauses falsified by the solution. Obviously, the cost of a solution found by an incomplete solver is greater or equal to the cost of the solution found by the complete solver: greater in case the incomplete solver could not find an optimum solution, equal otherwise;
- Solution Values: mandatory line starting with "v" followed by the truth assignment. The latter is the list of variables, each one eventually preceded by a – in case of negation.

In case of incomplete solver, the output may contain more "s", "o" and "v" lines to report the improvements in the solution. The input and output formats standardization enables UAQ-Solve to easily integrate any PMaxSAT solver participating in the competition as plug & play systems. In fact, it is enough to:

1. install the solver or however compile its code in the machine, and
2. specify the command line used to execute the solver in the main script of UAQ-Solve.

Consequently to the aforementioned rules, the first step for UAQ-Solve to solve a UAQ problem instance is to encode the clauses presented in Section 5.2 in Weighted Conjunctive Normal Form (WCNF). UAQ-Solve supports the encoding for both permission-based optimization, role-based optimization, and joint optimization. Listing 7.3 is the WCNF encoding of the instance presented in Listing 7.2 for permission-based optimization with $o_p = \min^4$.

Listing 7.3 WCNF encoding of the instance in Listing 7.2

```

1 p wcnf 15 41 41
2 41 -1 15 0
3 41 -2 10 0
4 41 -2 12 0
5 41 -2 14 0
6 41 -3 6 0
7 41 -3 7 0
8 41 -3 8 0
9 41 -3 9 0
10 41 -3 11 0
11 41 -3 13 0

```

⁴The other configurations can be inferred from Section 5.2 and are reported in Appendix B for the sake of completeness.

```

12 41 -3 14 0
13 41 -3 15 0
14 41 -4 7 0
15 41 -4 8 0
16 41 -4 9 0
17 41 -4 11 0
18 41 -4 12 0
19 41 -4 13 0
20 41 -5 6 0
21 41 -5 10 0
22 41 -6 3 5 0
23 41 -7 3 4 0
24 41 -8 3 4 0
25 41 -9 3 4 0
26 41 -10 2 5 0
27 41 -11 3 4 0
28 41 -12 2 4 0
29 41 -13 3 4 0
30 41 -14 2 3 0
31 41 -15 1 3 0
32 41 -2 -4 0
33 41 6 0
34 41 7 0
35 41 -8 0
36 1 -9 0
37 1 -10 0
38 1 -11 0
39 1 -12 0
40 1 -13 0
41 1 -14 0
42 1 -15 0

```

Row 1 is the parameters line: the UAQ problem reduced to PMaxSAT consists of $nbvar = 15$ variables, 5 for the roles and 10 for the permissions, and $nbclauses = 41$, clauses which will be explained below. The variables are labeled with integers from 1 to 15: in particular, variables 1 - 5 correspond to the roles (r_1, r_2, \dots, r_5), while variables 6 - 15 correspond to the permissions (p_1, p_2, \dots, p_{10}). The weight of the hard instances is ($top = nbclauses = 41$): this condition ensures the fulfillment of the requirement of having top greater than the sum of the weights of the soft clauses. Rows 2 - 31 are the hard clauses related to the Core RBAC, namely:

1. $(\neg \bar{r} \vee \bar{p})$ for all $p \in P$ and $r \in R_p$ (rows 2 - 21);

2. $(\neg \bar{p} \vee \bigvee \{\bar{r} : r \in R_p\})$ for all $p \in P$ (rows 22 - 31).

Row 32 corresponds to the encoding for the MER constraint SS-DMER($\{r2, r4\}, 2$). Here we use a naive way of converting the constraints like SS-DMER(rs, t), which consists in explicitly excluding all the possible combinations of t roles in rs being simultaneously activated, namely:

$$\bigwedge_{\substack{S \subseteq rs, \\ |S|=t+1}} \bigvee_{r \in S} \neg r$$

To this aim, we use the algorithm proposed in (26)⁵, which generates all the combinations of M elements drawn without replacement from a set of N elements. Unfortunately, this encoding requires $\binom{|rs|}{t+1}$ clauses, which in the worst case of $t = (|rs|/2 - 1)$ amounts to $O(2^{|rs|} / \sqrt{|rs|/2})$ clauses. This complexity bound is obtained through the Stirling's approximation of $n!$. A more elaborate result can be found in (81). However, also the smarter encoding proposed in (79)⁶ can be selected. This encoding is based on a parallel counter circuit designed by Muller and Preparata (65). Their counter recursively splits the input bits x_i (namely the variables for the roles in rs) into two halves and counts the number of 1 in each half. The results are then added using a standard binary adder. The result of the adder finally feeds a comparator, which checks whether the counter value is less than t . This encoding requires $7|rs| - 3\lfloor \log |rs| \rfloor - 6$ clauses and $2|rs| - 2$ auxiliary variables.⁷

Rows 33 - 35 are the clauses for the User Authorization Query, namely:

- a unit clause \bar{p} for each $p \in P_{lb}$ (p1 and p1, rows 33 and 34) and
- a unit clause $\neg \bar{p}$ for each $p \in P \setminus P_{ub}$ (p3, row 35);

Since the Core RBAC, the MER constraints, and the UAQ clauses are hard, they are all preceded by the weight 41.

Finally, rows 36 - 42 correspond to the clauses related to the extra permission optimization (minimization), namely:

⁵<http://www.netlib.no/netlib/toms/382>

⁶<http://www.carstensinz.de/software.html>

⁷ From some experimental results, it seems that the first encoding is still more convenient for problems with $o_p = \text{min}$. For $o_p = \text{max}$, the Sinz encoding is more convenient above all for benchmarks parametric in \hat{t} , for which the first encoding could actually explode. In the other cases, according to our experience, the difference in the solving time provided by the two encodings is not relevant in case of easy instances. It could be in the order of seconds or tens of seconds when the solving time is around 100 seconds. However, this bias is not such as to invalidate our experiments: except the case of benchmarks parametric in \hat{t} for $o_p = \text{max}$, we have never noticed a substantial difference in the behavior of the solvers. Obviously, the type of encoding to use deserves a further investigation when a real system must be implemented.

- a unit clause $\neg\bar{p}$ with weight set to 1 if $o_p = \text{min}$.

Since the latter are soft clauses, they are all preceded by the weight 1.

Once the problems are encoded in WCNF format like in Listing 7.3, UAQ-Solve can finally solve them exploiting one of the available solvers. At the time of writing, UAQ-Solve includes the following solvers:

- **LMHS⁸ (complete and incomplete versions):** LMHS is a MaxSAT solver implementing the implicit hitting set algorithm (50) for MaxSAT (34; 50). After a first preprocessing through MaxPre (82) which simplifies the problem, the MaxSAT cost function is given as input to CPLEX 12.7 (45), which is an optimizer, while the CNF formula (hard clauses and soft clauses) are given as input to MiniSat 2.2 (37), which is used as satisfiability checker. In the implicit hitting set loop, MiniSat checks the satisfiability of the formula excluding a hitting set in order to find an unsatisfiable core. Unsatisfiable cores are then accumulated in a set, for which the CPLEX finds a minimum-cost hitting set with reference to the cost function. LP-based reduced-cost fixing techniques for MaxSAT (38) together with bounds enable to simplify the problem by hardening or relaxing some soft clauses during the search. While the optimizer proves the lower bounds, the upper bounds on the optimal solution cost are found during core minimization procedure and non-optimal hitting set phase;
- **Loandra⁹ (complete and incomplete versions):** Loandra is a MaxSAT solver whose architecture consists of two closely interleaved parts: the preprocessor and the solver. The former is MaxPre (45), modified to support the addition of clauses, while the latter is a reimplementation of the PMRES MaxSAT algorithm (67) extended with weight-aware core extraction as described in (19). First, Loandra preprocesses the input instance, which is then given as input to the solver. Loandra makes extensive use of SAT-based preprocessing using labels (1; 2): each soft clause is first extended with a fresh variable, then the preprocessor is invoked on the clauses in $F_h \cup \{C \vee l_C \mid C \in F_s\}$, where F_h and F_s are the sets of hard and soft clauses respectively and l_C is the fresh variable introduced for the soft clause C . During the preprocessing phase, MaxPre is forbidden from resolving on the added labels. The preprocessed instance is then converted back to standard MaxSAT: all the clauses, including the soft ones, are treated as hard and a soft clause $\neg l_C$ is introduced for each fresh variable introduced in the previous phase. Each soft clause has the same weight of the corresponding original

⁸<https://github.com/psaikko/LMHS>

⁹<https://github.com/jezberg/loandra>

clause for which the fresh variable was introduced. Actually, during the initialization of the solver, the soft clauses of the form $\neg l_C$ are not introduced; instead, the literals are reused as assumption variables to be used in core extraction. The preprocessed instance is then provided as input to the solver. Loandra also leverages stratification and clause hardening (8), clause cloning through assumptions and reusing assumption variables as relaxation variables (19). During the solver execution, all cardinality constraints added due to core relaxation are also added to the preprocessor as well. When the working formula is sufficiently modified, the execution is switched back to the preprocessor. The latter tries to further simplify the working formula: in case of success, the solver is reinitialized again on the simplified formula. Once the solver terminates, the preprocessor rebuilds the optimal model for the original formula and finally Loandra terminates the execution;

- **MaxHS¹⁰ (complete and incomplete versions):** MaxHS (33) is a MaxSat solver that leverages the Implicit Hitting Set approach, using CPLEX 12.7 (45) as Integer Programming solver and minisat 2.2¹¹ as SAT solver in a hybrid approach to MaxSat solving. Using an LP relaxation and the reduced costs associated with the optimal LP solution, MaxHS can harden or immediately falsify some soft clauses, as better explained in (38). In addition, when sets of soft clauses having the same weight of which at most one can be falsified or at most one can be satisfied are detected, the soft clauses themselves can be more compactly encoded with a single soft clause to better exploit such information. MaxHS maintains an upper bound and a lower bound, which are the costs related to the best model found and the optimal solution respectively. CPLEX is responsible for providing valid lower bounds. When the gap between the lower bound and upper bound is low enough, the upper bound model corresponds to the optimal solution. Consequently, MaxHS terminates the execution;
- **Maxino¹² (complete solver):** Maxino is a MaxSAT solver built on top of the SAT solver Glucose 4.1 (24). Besides, Maxino implements the algorithm K, which is based on k-ProcessCore (54). The latter is in turn a parametric algorithm generalizing OLL (15), ONE (54), and PMRES (67). In a first moment, the MaxSAT instance undergoes a process known as relaxation, in which non-unary soft clauses are replaced with fresh variables. In particular, the soft clause C is replaced with the clause $C \vee \neg x$, where x is the fresh variable introduced for C . During the relaxation, the weight associated with

¹⁰<https://github.com/fbacchus/MaxHS>

¹¹<http://minisat.se/MiniSat.html>

¹²<https://alviano.net/software/maxino>

the soft clause is associated with the corresponding fresh variable. After the relaxation, the normalized problem consists of hard clauses and soft literals. Consequently, the computational problem amounts to maximize the linear function defined by the soft literals, which are subject to the set of hard clauses. With this definition, a model of the constraints that satisfies all the soft literals is an optimum model. Maxino tackles the problem by leveraging the unsatisfiable core analysis. If during the search of the optimum model an inconsistency arises, the unsatisfiable core returned by the SAT solver is analyzed. The analysis of unsatisfiable core is preceded by a shrink procedure (7) with the aim of reducing the size of the unsatisfiable core, even if the procedure does not necessarily returns an unsatisfiable core of minimal size. After the analysis of the unsatisfiable core, ONE, OLL, PMRES and K introduce a set of new constraints making use of new soft literals, which replace the soft literals involved in the unsatisfiable core. The bound used in the constraints introduced by K is calculated starting from k , which is dynamically determined based on the size of the analyzed unsatisfiable core. Finally, the solver executes over the modified problem. This process can be iterated until there is no model that satisfies all replaced soft literals, namely until the optimum model is found;

- **Open-WBO¹³ (complete and incomplete versions):** Open-WBO is a MaxSAT solver that implements a variety of algorithms for solving MaxSAT and Pseudo-Boolean (PB) formulas. All the algorithms are based on a sequence of calls to a MiniSAT-like solver (37). Open-WBO has been one of the best solvers in the MaxSAT Evaluations of for different years. Different versions of Open-WBO do exist: in our work we make use of Open-WBO-LSU and Open-WBO-RES. The former is based on a linear search algorithm SAT-UNSAT (20) with lexicographical optimization for weighted problems (48). Open-WBO-LSU performs a sequence of calls to a SAT solver, while refining an upper bound on the number of unsatisfied soft clauses. In order to restrict the upper bound at each iteration, a cardinality constraint must be encoded into CNF for unweighted problems, while a pseudo-Boolean constraint must be encoded into CNF for weighted problems. In the first case, Open-WBO-LSU leverages the Modulo Totalizer encoding (83), in the second one, it uses the Generalized Totalizer encoding (GTE) (72). Open-WBO-RES is based on the unsatisfiability-based algorithms MSU3 (60) and OLL (3), which work by iteratively refining a lower bound on the number of

¹³<https://github.com/sat-group/open-wbo>

unsatisfied soft clauses until an optimum solution is found. Both MSU3 and OLL use the Totalizer encoding for incremental MaxSAT solving (70);

- **QMaxSAT and QMaxSATuc¹⁴ (complete and incomplete versions):** QMaxSAT is a SAT-based MaxSAT solver which uses CNF encoding of Pseudo-Boolean (PB) constraints (56). The version leveraged in UAQ-Solve is an adjustment of the solver Glucose 3.0 (14; 37). QMaxSAT implements a model-guided algorithm, in which a blocking variable is added to each soft clause. Solving the MaxSAT problem is then reduced to find the SAT model that minimizes $\sum_{i=1}^m w_i \cdot b_i$, where w_i is the weight of the soft clause C_i and m is the number of soft clauses involved in the MaxSAT problem. The PB constraints $\sum_{i=1}^m w_i \cdot b_i < k$ are manipulated by Glucose, which encodes them into SAT. According to k and the sum of the weights of the soft clauses, one of the following encodings is used to encode the PB constraints: Totalizer (17), Binary Adder (89), Modulo Totalizer (83), and Weighted Totalizer (44), and Mixed Radix Weighted Totalizer (66). QMaxSATuc is a hybrid solver that follows and alternates both a core-guided and a model-guided mode. In core-guided mode, the blocking variables are negated and then passed as assumptions to Glucose, which treats each literal as a unit clause. Glucose returns a subset of assumptions used in the UNSAT proof. Each soft clause corresponding to a blocking variable in the subset constitutes an element in the unsat-core. Then, a clause having all blocking variables in the subset as literals is created and added to the clause database in order to eliminate the core. In addition, all the blocking variables in the subset are subtracted from the set of blocking variables. In model-guided mode, nothing is passed to Glucose as assumptions, like in QMaxSAT;
- **WPM1-2012¹⁵ (complete solver):** WPM1 (24) is an extension to the Weighted Partial MaxSAT problem of the Fu and Malik algorithm (42; 92), which was originally designed for Partial MaxSAT. The Fu and Malik algorithm consists in iteratively calling a SAT solver on a working formula. In case the latter is unsatisfiable, the SAT solver provides an unsatisfiable core. The algorithm creates a new blocking variable for each soft clause in the unsatisfiable core. Then a new working formula is produced by adding the new blocking variables to the soft clauses of the core and introducing a cardinality constraint saying that exactly one of the new variables should be true. Finally, the counter of falsified clauses is incremented by one. This procedure is repeated until the working formula is satisfiable. WPM1 is the weighted version

¹⁴<https://sites.google.com/site/qmaxsat/>

¹⁵http://ulog.udl.cat/?page_id=30

of the algorithm just described and is built on top of the SAT solver picosat v.924 (21). The latter is iteratively called with a weighted working formula, but excluding the weights. Differently from the Fu and Malik algorithm, when the SAT solver returns an unsatisfiable core, the working formula is transformed by duplicating the clauses of the core. In one of the copies, the clauses have the original weight minus the minimum weight of the clauses of the unsatisfiable core; in the other copy, the blocking variables are introduced with the minimum weight. Then the cardinality constraint on the blocking variables is created as in the Fu and Malik algorithm. The cardinality constraints introduced by WPM1 are translated into SAT through the regular encoding (23). The latter ensures a linear complexity on the size of the cardinality constraint. Finally, the cost is incremented by the minimum weight of the clauses of the unsatisfiable core. The WPM1 solver included in UAQ-Solve actually applies the stratified approach, an improvement that consists in restricting the set of clauses sent to the SAT solver to force it to concentrate on those with higher weights. Consequently, the SAT solver returns unsatisfiable cores with clauses with higher weights. Once the SAT solver returns SAT, it is allowed to use clauses with lower weights. This expedient contributes to increase the cost faster (25).

While launching the execution of UAQ-Solve, the user should specify one or more solvers to use as an option in the command line. According to the solver specified, the proper command line is selected and executed to run the solver. However, if the latter is not specified a default solver is used¹⁶. Since the execution of the solvers over particularly hard instances can consume many resources, UAQ-Solve executes the instances sequentially to avoid crashes.

Since we do not want to go into the details of the algorithms underlying the solvers and their implementation, we treat the solvers as black boxes. As a consequence, UAQ-Solve executes them with the default configurations set by the developers. It is obvious that, knowing the underlying algorithms, one could find better configurations that could lead also to better performance¹⁷. It is then important to point out that, for this reason, we do not have any guarantee that the experimental results shown in Chapter 8 correspond to the actual best performance of the solvers.

The aforementioned solvers have been included in UAQ-Solve because of their placement at the MaxSAT Evaluation 2017 and 2018, which are summarized in Tables 7.1 (complete tracks) and 7.2 (incomplete tracks). We also included WPM1-2012, a complete solver which

¹⁶In our configuration, the default solver, but it can be easily changed.

¹⁷The configuration that could lead to the best performance could also vary from problem to problem. However, this is out of scope in this thesis.

was ranked first for the Crafted category and second for the Industrial category at the MaxSAT Evaluation 2012. However, as already stated before, it is easy to integrate other solvers in the tool. In the future, we will include other solvers that participated in the competition in 2018 and 2019.

Table 7.1 Solvers placement at MaxSAT Evaluation 2017 and 2018 for Complete Tracks

Solver	Unweighted	Weighted
	Complete Track	Complete Track
MaxHS	2nd in 2017	1st in 2017 3rd in 2018
Maxino	3rd in 2017 3rd in 2018	
Open-WBO-RES	1st in 2017	
QMaxSAT		2nd in 2017
QMaxSATuc		3rd in 2017

Table 7.2 Solvers placement at MaxSAT Evaluation 2017 and 2018 for Incomplete Tracks

Solver	Unweighted	Unweighted	Weighted	Weighted
	Incomplete Track (60 s)	Incomplete Track (300 s)	Incomplete Track (60 s)	Incomplete Track (300 s)
LMHS-inc			2nd in 2017	3rd in 2017
MaxHS-inc	2nd in 2017	3rd in 2017		2nd in 2017
Open-WBO-LSU	1st in 2017	2nd in 2017	3rd in 2017	

Since any complete solver runs until it finds an optimum solution, we make use of the software runsolver¹⁸ to force a timeout: if the solver can not find an optimum solution before the timeout, a SIGTERM and then a SIGKILL signals are sent and the UAQ problem instance is labelled as "skipped". The default value for the timeout is 10 minutes, but it can be modified providing the desired values as argument.

The output of UAQ-Solve is a set of files containing some statistics on the experiments, in particular:

¹⁸<http://www.cril.univ-artois.fr/~roussel/runsolver>

- A file containing the details of the encoding for any UAQ instance: the encoding time, the number of variables and the number of clauses¹⁹;
- A file containing the mean encoding time and the mean number of variables and clauses for every step of the dimension under examination;
- A file containing the solving details for any UAQ problem instance: the number of variables and clauses in the WCNF encoding, the solving time and the status (OPTIMUM, BEST in case of incomplete solver, UNSAT and SKIPPED);
- A file containing the mean number of variables and clauses of the WCNF encoding, the mean and median solving time and the percentage of skipped problems for every step of the dimension under examination;
- A file containing the quartiles of the solving time for every step of the dimension under examination.

For the sake of completeness, in Listings 7.4, 7.5, 7.6, 7.7 and 7.8 we report the content of the five output files for the benchmark generated by UAQ-Solve with the specification file in Listing 7.1.

Listing 7.4 Summary encoding file for the benchmark generated with Listing 7.1

	RPP	#MERS	MERB	RPC	Filename	T	V	C
2	2	1	2	2	ss-d-5-2-0-MIN.uaq	0.004	15	41
2	2	1	2	2	ss-d-5-2-1-MIN.uaq	0.004	15	41
2	2	1	2	2	ss-d-5-2-2-MIN.uaq	0.004	15	41
2	2	1	2	2	ss-d-10-2-0-MIN.uaq	0.004	20	41
2	2	1	2	2	ss-d-10-2-1-MIN.uaq	0.004	20	41
2	2	1	2	2	ss-d-10-2-2-MIN.uaq	0.004	20	41
2	2	1	2	2	ss-d-15-2-0-MIN.uaq	0.004	25	41
2	2	1	2	2	ss-d-15-2-1-MIN.uaq	0.008	25	41
2	2	1	2	2	ss-d-15-2-2-MIN.uaq	0.004	25	41

Listing 7.5 Plot encoding file for the benchmark generated with Listing 7.1

	RPP	#MERS	MERB	RPC	NR	NP	P1b	TMed	TMean	NV	NC	DIMACS	ERRS
2	2	1	2	2	5	10	2	0.004	0.004	15	41	3	0
2	2	1	2	2	10	10	2	0.004	0.004	20	41	3	0
2	2	1	2	2	15	10	2	0.004	0.005	25	41	3	0

¹⁹The interested reader can find details on the encoding of our benchmark in Appendix C.

Listing 7.6 Summary file for the benchmark generated with Listing 7.1

	RPP	#MERS	MERB	RPC	Filename	T	V	C	Res
2	2	1	2	2	ss-d-5-2-0-MIN.dimacs	0.000	15	41	UNSAT
3	2	1	2	2	ss-d-5-2-1-MIN.dimacs	0.000	15	41	OPTIMUM
4	2	1	2	2	ss-d-5-2-2-MIN.dimacs	0.000	15	41	UNSAT
5	2	1	2	2	ss-d-10-2-0-MIN.dimacs	0.000	20	41	OPTIMUM
6	2	1	2	2	ss-d-10-2-1-MIN.dimacs	0.000	20	41	OPTIMUM
7	2	1	2	2	ss-d-10-2-2-MIN.dimacs	0.000	20	41	OPTIMUM
8	2	1	2	2	ss-d-15-2-0-MIN.dimacs	0.000	25	41	OPTIMUM
9	2	1	2	2	ss-d-15-2-1-MIN.dimacs	0.000	25	41	OPTIMUM
10	2	1	2	2	ss-d-15-2-2-MIN.dimacs	0.000	25	41	OPTIMUM

Listing 7.7 Plot file for the benchmark generated with Listing 7.1

	RPP	#MERS	MERB	RPC	NR	NP	Plb	TMed	TMean	NV	NC	PUNSAT	PSK	DIMACS	ERRS
2	2	1	2	2	5	10	2	0.000	0.000	15	41	0.67	0	3	0
3	2	1	2	2	10	10	2	0.000	0.000	20	41	0	0	3	0
4	2	1	2	2	15	10	2	0.000	0.000	25	41	0	0	3	0

Listing 7.8 Quantiles file for the benchmark generated with Listing 7.1

	RPP	#MERS	MERB	RPC	NR	NP	Plb	B1-H	B2-L	B3-U	W_t	W_b
2	2	1	2	2	5	10	2	0	0	0	0	0
3	2	1	2	2	10	10	2	0	0	0	0	0
4	2	1	2	2	15	10	2	0	0	0	0	0

UAQ-Solve can also be used to compare the performance of different solvers providing the solvers to compare and the timeout. When used to compare complete solvers, for each UAQ problem instance, UAQ-Solve provides the solving time; when used to compare incomplete solvers, UAQ-Solve provides the cost of an optimum solution found by a complete solver and the cost of the solutions found by the incomplete solvers. This is an interesting piece of information, because it gives an idea of the quality of the best solutions found by the incomplete solvers related to the optimum ones.

For example, Listing 7.9 presents the results of the comparison of the complete solvers Loandra, LMHS and MaxHS over the benchmark generated with the specification file in Listing 7.1. The benchmark is really simple, thus here there is not a clear difference in the solving times.

Listing 7.9 Solving times of different complete solvers over the benchmark generated with Listing 7.1

	RPP	#MERS	MERB	RPC	FILE	Loandra	LMHS	MaxHS
2	2	1	2	2	ss-d-5-2-0-MIN.dimacs	0	0.004	0.004
3	2	1	2	2	ss-d-5-2-1-MIN.dimacs	0	0.004	0
4	2	1	2	2	ss-d-5-2-2-MIN.dimacs	0	0.004	0
5	2	1	2	2	ss-d-10-2-0-MIN.dimacs	0	0.004	0
6	2	1	2	2	ss-d-10-2-1-MIN.dimacs	0	0.004	0
7	2	1	2	2	ss-d-10-2-2-MIN.dimacs	0	0.004	0
8	2	1	2	2	ss-d-15-2-0-MIN.dimacs	0	0.004	0
9	2	1	2	2	ss-d-15-2-1-MIN.dimacs	0	0.004	0
10	2	1	2	2	ss-d-15-2-2-MIN.dimacs	0	0.004	0

In the same way, we show the results of the comparison of the incomplete solvers Loandra-inc and LMHS-inc. The -1 in rows 2 and 4 are due to the fact that the related instances are unsatisfiable. Since the other instances are very easy to solve, the incomplete solvers found the optimum solution before the timeout and, consequently, the costs of the solutions found by both Loandra-inc and LMHS-inc are the same as the ones of the complete solver.

Listing 7.10 Solution costs over the benchmark generated with Listing 7.1

	RPP	#MERS	MERB	RPC	FILE	OPT	Loandra-inc	LMHS-inc
2	2	1	2	2	ss-d-5-2-0-MIN.dimacs	-1	-1	-1
3	2	1	2	2	ss-d-5-2-1-MIN.dimacs	5	5	5
4	2	1	2	2	ss-d-5-2-2-MIN.dimacs	-1	-1	-1
5	2	1	2	2	ss-d-10-2-0-MIN.dimacs	2	2	2
6	2	1	2	2	ss-d-10-2-1-MIN.dimacs	1	1	1
7	2	1	2	2	ss-d-10-2-2-MIN.dimacs	0	0	0
8	2	1	2	2	ss-d-15-2-0-MIN.dimacs	0	0	0
9	2	1	2	2	ss-d-15-2-1-MIN.dimacs	0	0	0
10	2	1	2	2	ss-d-15-2-2-MIN.dimacs	0	0	0

Chapter 8

Experimental results

This chapter describes the major results of our work. In particular:

Section 8.1 demonstrates that state-of-the-art benchmarks are incomplete and unsatisfactory.

In fact, as predicted by our methodology, they only consists of instances that never stimulate an exponential behavior. According to our methodology, only the benchmark parametric in $|P_{lb}|$ in principle could stimulate that behavior. UAQ-Solve can solve all the instances contained in the benchmarks (even the one parametric in $|P_{lb}|$) in a fraction of a second, as we will show in Figure 8.1. The latter shows also that UAQ-Solve outperforms the solvers 2D-Opt-Search and 2D-Opt-CNF over the benchmarks. Another issue is that all the benchmarks were designed for $o_p = \min$, thus benchmarks for $o_p = \max$ are totally missing;

Section 8.2 contains the evaluation of our benchmarks designed through our methodology for $o_p = \min$. The benchmarks are evaluated by running UAQ-Solve over them. All the benchmarks proposed stimulate the behavior for which they were designed. We conclude that the suite of benchmarks designed for $o_p = \min$ is empirically adequate. The set of benchmarks consists of:

- Two benchmarks parametric in $|P_{lb}|$ having different values of $|R|$. One of them is solved in exponential time; all the instances of the other benchmark are always solved in around 1 to 6 seconds regardless of the value of $|P_{lb}|$ (see Figure 8.4a);
- Two benchmarks parametric in $|R|$ having different values of $|P_{lb}|$. Similarly to the previous case, one benchmark is solved in exponential time; all the instances of the other benchmark are solved in a fraction of a second, regardless the value of $|R|$ (see Figure 8.5a);

- Three benchmarks parametric in \widehat{R}_P having different value of $|P_{lb}|$. All the benchmarks are solved in polynomial time. These benchmarks highlight the influence of $|P_{lb}|$, which, according to our methodology, is the degree of the polynomial (see Figure 8.6a);
- Four benchmarks, each of them parametric in one of the remaining dimensions, namely $|P_{ub}|$, $|C|$, \widehat{rs} , and \widehat{t} . Over all these benchmarks, the increase in the solving time is not relevant (see Figure 8.7).

Figure 8.8 will show the comparison of the performance of UAQ-Solve over some of the benchmarks encoded for permission-based and joint optimization. The experiment shows that joint optimization is not particularly detrimental for performance. Finally, we evaluate 2D-Opt-Search and 2D-Opt-CNF over the benchmarks. UAQ-Solve outperforms them for the majority of the instances (see Figure 8.1);

Section 8.3 contains the evaluation of our benchmarks designed through our methodology for $o_p = max$. The benchmarks are evaluated by running UAQ-Solve over them. All the benchmarks proposed but one stimulate the behavior for which they were designed. We conclude that the suite of benchmarks designed for $o_p = max$ is empirically adequate. The set of benchmarks consists of:

- Two benchmarks parametric in $|R|$ having different values for $|C|\widehat{t}$. One of them is solved in exponential time, while the other is solved in almost constant time (see Figure 8.16);
- One benchmark parametric in $|P_{ub}|$. Contrary to our expectation, UAQ-Solve solves this benchmark in exponential time (see Figure 8.17). We should better investigate in the future;
- Two benchmarks parametric in $|C|$, having different values of $|R|$. One of them is solved in exponential time, while the other is solved in almost constant time (see Figure 8.18);
- Two benchmarks parametric in \widehat{t} , having different values of $|R|$. One of them is solved in exponential time; all the instances contained in the other benchmark are solved in a fraction of second (see Figure 8.19);
- Three benchmarks parametric in \widehat{rs} having different values of $|C|\widehat{t}$. All the benchmarks are solved in polynomial time. These benchmarks highlight the influence of $|C|\widehat{t}$, which, according to our methodology, is the degree of the polynomial (see Figure 8.20);

- One benchmark parametric in \widehat{R}_P and one benchmark parametric in $|P_{lb}|$. All the instances are solved in a fraction of a second, regardless of the value of the parameter under examination.

Figure 8.22 will show that the performance of UAQ-Solve deteriorates with joint optimization;

Section 8.4 compares the performance of different PMaxSAT solvers over some of our benchmarks. The experimental results show that:

1. Focusing on a singular dimension, the solver that best performs on small values of the parameter may not be the best also for big values of the same parameter, see Maxino over P_{lb_bigR} in Figure 8.27a;
2. The solver that best performs over a certain class of UAQ problems could not be the best also for the other classes. For example, in Figure 8.27a, for high values of $|P_{lb}|$, MaxHS is the best solver among the ones under test, while, in Figure 8.29a, the best solver is Maxino;
3. In general, the solvers exhibit the same behavior, however it may happen that a solver clearly outperforms the others, see Maxino over $that_bigR$ in Figure 8.33a.

Section 8.5 shows the results of the execution of incomplete solvers over our benchmarks. From the experiments it appears that:

- The cost of the best solution found comes closer to the cost of the optimum solution with the increase of the timeout (see Figure 8.34a);
- The incomplete solver seems to be able to quickly find a good solution, then it takes more time to converge to the optimum one (see Figure 8.35a);
- Considering the time spent by any complete solver to find the optimum solution, the error ε of the solution found by an incomplete solver in 1 second is acceptable (see Figure 8.36b);
- It may happen that any incomplete solver finds a solution that minimizes the solution cost, but that it does not have enough time to prove that it actually is an optimum solution;
- Although any incomplete solver could provide a sub-optimal solution, in terms of solution cost (namely, in case of $o_p = \text{min}$, safety), the use of any incomplete solver is still more convenient than using a complete solver with $o_p = \text{any}$ (see Figure 8.36e);

- Over hard benchmarks and with high timeouts (such as 600 seconds), the incomplete solvers could be less performing than the complete ones (see Figure 8.37);
- The performance of a complete and an incomplete solver seem to be comparable when executing over easy benchmarks (see Figure 8.38).

The experiments have been conducted on a PC with 2 64-bit Intel Xeon CPU X7350 (8 core) @ 2.93GHz and 47 GB RAM running Linux (Ubuntu 16.04.5 LTS). For every experiment, we set the timeout to 600 seconds. After this threshold, the instance under test was labelled as "SKIPPED".

In the following chapter we do not provide any information on the encoding time and the number of variables and clauses for a number of reasons:

- The time spent by UAQ-Solve to encode the benchmarks in WCNF is not particularly relevant;
- In principle, the RBAC policy could be encoded once. At run-time it would be enough to add to the encoding the clauses related to the DMER constraints¹ and to the query. This would further decrease the time spent for getting the WCNF encoding;
- The dimension of the encodings follows the complexity results presented in Section 5.2²;
- In earlier research, we thought that the problem complexity could depend only on the number of variables and clauses. However, we have evidence that this does not hold³.

We also point out that when we talk about the solving time of UAQ-Solve, we do not include the encoding time.

The reader interested in more details on the encoding can find them in Appendix C.

8.1 Benchmarks from (64)

In Section 8.1.1, we leverage the methodology introduced in Section 6.1 to evaluate the benchmarks from (64) and presented in Table 6.4 by running UAQ-Solve, leveraging the PMaxSAT solver Loandra. The result of the evaluation is that the suite of benchmarks is

¹If MS-DMER, SS-HMER and MS-HMER constraints are used, they must be properly reduced to SS-DMER constraints, see Section 4.4

² $O(|R||P|)$ clauses and $O(|R| + |P|)$

³See Appendix C

incomplete and unsatisfactory, because it consists only of easy benchmarks. In fact, our methodology predicted that the solving time of UAQ-Solve over four of the five benchmarks would not have increased in a relevant way. The only benchmark that in principle could exhibit an exponential growth is the one parametric in P_{lb} . However, all its instances are solved in less than 10 millisecond, thus meaning that the benchmark is empirically inadequate. In addition, the benchmarks were designed only for $o_p = \min$.

Secondly, in Section 8.1.2, we use the same benchmarks to experimentally evaluate and compare the performance of the following solvers:

- 2D-Opt-Search (64): a search-based solver leveraging the FPP result;
- 2D-Opt-CNF (64): a SAT-based solver that leverages the reduction of the UAQ Decision Problem to SAT, zChaff⁴ as SAT solver a state-of-the-art SAT solver and a two-dimensional binary search to solve UAQ problems;
- UAQ-Solve.

The reason why we decided to compare UAQ-Solve with 2D-Opt-Search and 2D-Opt-CNF is that they are proposed by the same authors as the benchmarks and the code is available online⁵. The results show that UAQ-Solve outperforms both 2D-Opt-search and 2D-Opt-CNF over all the benchmarks.

8.1.1 Benchmarks evaluation

In the evaluation step, we run UAQ-Solve over the benchmarks instances encoded with simple permission optimization, namely with $o_p = \min$, $o_r = \text{any}$ and $pri = p$. The pink lines in Figure 8.1 represent the results of the evaluation of the benchmarks. In particular, Figure 8.1a shows the median solving time of UAQ-Solve over the *roles* benchmark. The value of $|R|$ ranges from 25 to 200, however the median solving time never exceeds 4 milliseconds. This result confirms the prediction we presented in Section 6.2: in fact, the value of the other UAQ problem dimensions makes Algorithm 3 preferable to Algorithm 1.

Figures 8.1b, 8.1c, and 8.1d present the median solving time of UAQ-Solve over the benchmarks related to the MER constraints features, namely d , *rolesPerConstr*, and t . In the related benchmarks, both $|C|$ and $\hat{r}s$ vary from 10 to 100, while \hat{t} varies from 2 to 12. The plots present a slight increase on varying of C and $\hat{r}s$, whose maximum values are 8 milliseconds and 16 milliseconds respectively. Instead, the median solving time is a constant

⁴<http://www.princeton.edu/~chaff/zchaff.html>

⁵<https://ece.uwaterloo.ca/~tripunit/uaq/>.

value of 4 milliseconds on varying of \hat{t} . Also these results confirm our expectation: in fact, for $o_p = \min$ the algorithms presented in Section 5.1 are insensitive to the variation of the parameters under discussion.

Finally, 8.1e shows the experimental results obtained by running UAQ-Solve against the *plib* benchmark. Contrary to our expectation, UAQ-Solve could solve all the problems contained in the benchmark in few milliseconds (at most 4) even for largest value of $|P_{lb}|$ (11), while it should have exhibited an exponential growth. This result means that the benchmark *plib* is experimentally inadequate.

We can thus conclude that the set of benchmarks provided by (64) is incomplete and unsatisfactory: in particular, it consists only of instances that can be used to check the solvers' performance over easy UAQ problems.

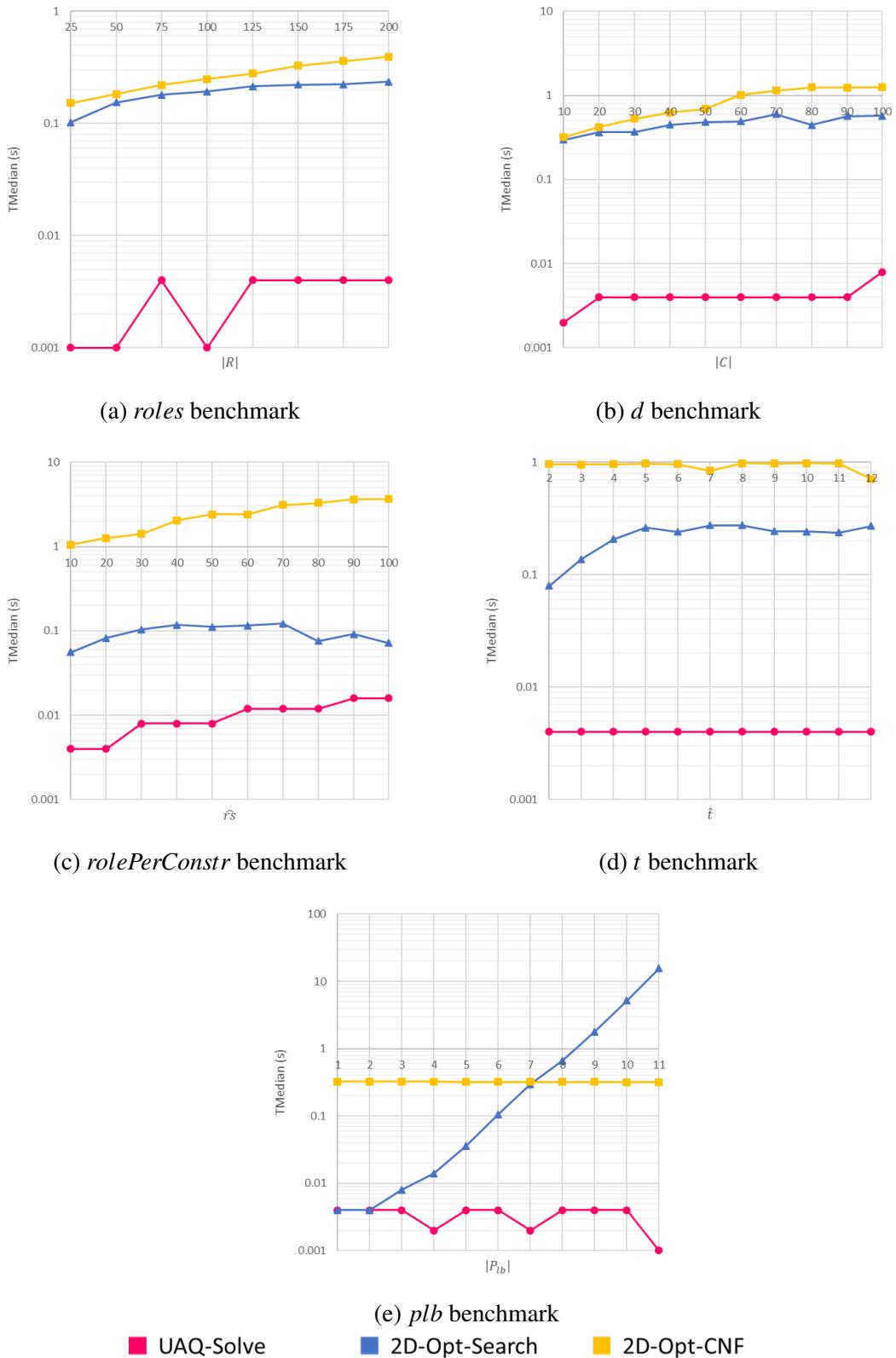


Figure 8.1 Median solving time of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks presented in Table 6.4 with permission-based optimization

8.1.2 Solvers evaluation

After the evaluation provided in the previous subsection, we know that the benchmarks under test consist of easy instances of UAQ problem. We can then use them to assess the solvers' performance: here we expect that any reasonably efficient solver solve them quickly. If this is not the case, we conclude that the solver under evaluation is empirically inefficient against the particular family of UAQ problem. In addition to the curves already discussed in Subsection 8.1.1, Figure 8.1 shows the performance of the solvers 2D-Opt-Search and 2D-Opt-CNF over the examined benchmarks.

It is immediate to note that UAQ-Solve always outperforms both 2D-Opt-Search and 2D-Opt-CNF for all the benchmarks. Apart from *roles*, UAQ-Solve and 2D-Opt-CNF exhibit similar behavior, however there is always a distance of two orders of magnitude between the median solving time of the two solvers. Instead, the median solving time of 2D-Opt-CNF over *roles* slowly increases with the increase in $|R|$, while UAQ-Solve always takes 1 or 4 milliseconds. When running over the benchmarks *roles*, $|C|$, \hat{rs} , and \hat{t} , 2D-Opt-search always outperforms 2D-Opt-CNF. In particular, for *rolesPerConstr* there is a distance of at least one order of magnitude between the median solving time of 2D-Opt-search and 2D-Opt-CNF. On the contrary, for *plib*, the shapes of the curves representing the median solving time are completely different: while 2D-Opt-CNF always solves the instances in 320 milliseconds regardless of the value of $|P_{lb}|$, 2D-Opt-Search exhibits an exponential growth, from 4 milliseconds when $|P_{lb}| = 1$ to more than 15 seconds when $|P_{lb}| = 11$.

In Figure 8.2 we present the results of the execution of the solvers over the benchmarks encoded with joint optimization, namely with $o_p = \min$, $o_r = \min$ and $pri = p$. Also in this case, UAQ-Solve solves all the problems in a fraction of second, outperforming both 2D-Opt-Search and 2D-Opt-CNF. However, contrary to the case with permission-based optimization, 2D-Opt-Search always performs worse than 2D-Opt-CNF. Except for *plib*, the curves representing the median solving time of UAQ-Solve and 2D-Opt-Search is similar, but UAQ-Solve always performs three orders of magnitude better. Instead, the behavior of the two solvers is different for *plib*: the median solving time of UAQ-Solve is 8 milliseconds for every value of $|P_{lb}|$ of which we have evidence; on the contrary, the median solving time of 2D-Opt-Search grows from 5 to 162 seconds with $|P_{lb}|$ ranging from 1 to 11. 2D-Opt-CNF over *roles* and *d* shows the same behavior as UAQ-Solve, but the latter always performs two orders of magnitude better. Over the other benchmarks, the median solving time of 2D-Opt-CNF is fluctuating, however always at least one order of magnitude higher compared to UAQ-Solve. For the sake of completeness, in Figure 8.3 we report the difference between

the performance of the three solvers with joint and permission-based optimization, namely

$$\Delta_{TMedian} = TMedian_{joint} - TMedian_{permission-based}.$$



Figure 8.2 Median solving time of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks presented in Table 6.4 with joint optimization

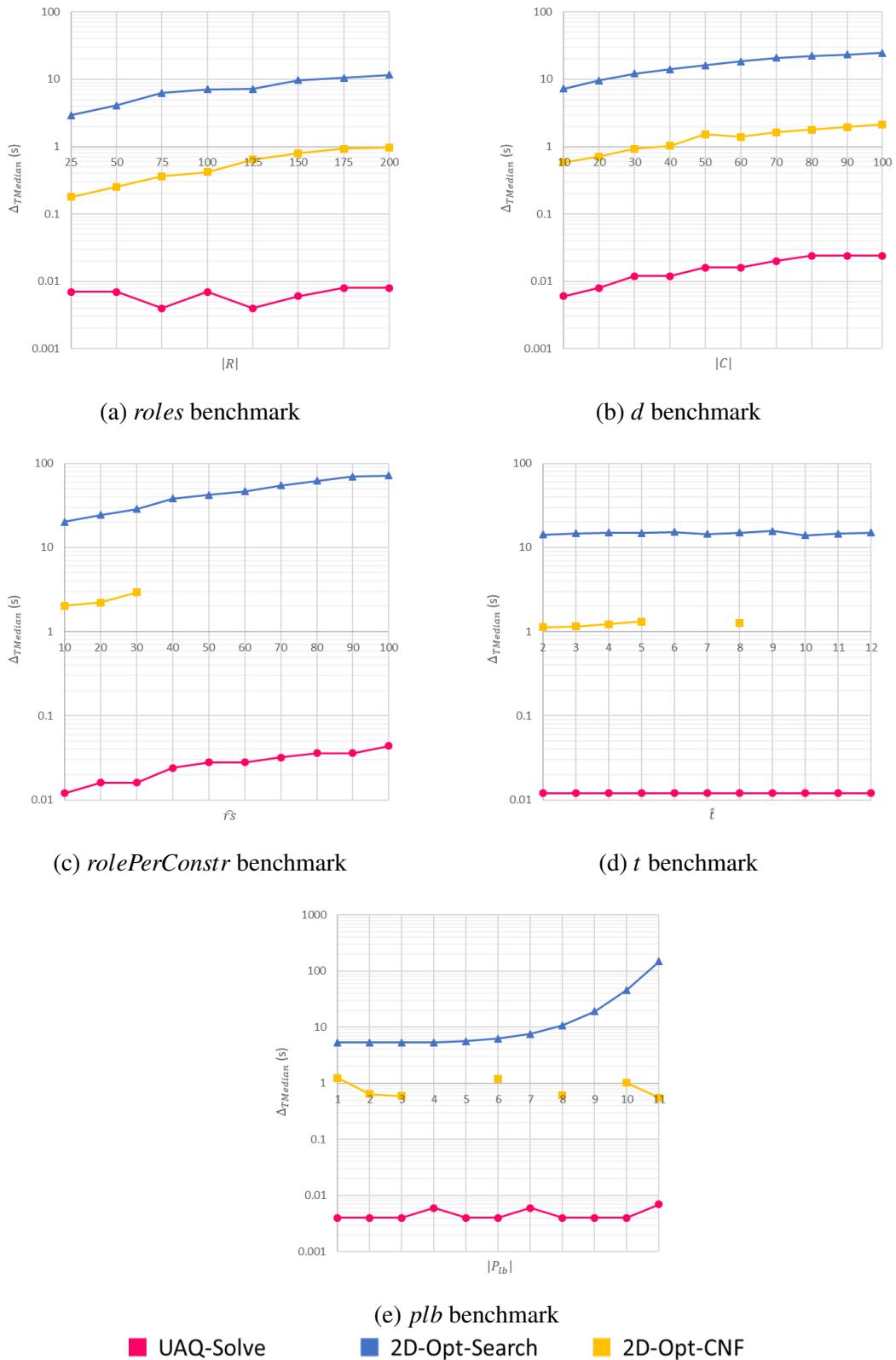


Figure 8.3 ΔT_{Median} of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks presented in Table 6.4

The impact of joint optimization on the performance of UAQ-Solve is a fraction of second. In the plots related to $rolesPerConstr$, \hat{t} , and $|P_{lb}|$, some points related to the median solving time of 2D-Opt-CNF are missing, because for those values of the parameter under examination 2D-Opt-CNF performs better on the instances encoded with joint optimization, rather than the ones with permission-based optimization. However, the maximum value of $Delta_{TMedian}$ is 2.215 seconds.

The joint optimization mainly impacts on the performance of 2D-Opt-Search: in fact, for $roles$, d , and $rolesPerConstr$, $Delta_{TMedian}$ increases respectively from 2.908 to 11.624 seconds with $|R|$ ranging from 25 to 200, from 7.247 to 24.59 seconds with $|C|$ ranging from 10 to 100, from 20.084 to 71.303 seconds with $\hat{r}s$ ranging from 10 to 100. For t , $Delta_{TMedian}$ is between 14 and 16 seconds for each value of \hat{t} for which we have evidence. Finally, for plb , $Delta_{TMedian}$ is less than 10 seconds when $|P_{lb}| \leq 10$, then it rapidly increases from 10.702 to 147.196 seconds.

8.2 Our benchmarks with $o_p = \min$

In this section, we repeat the same operations we described in Section 8.1 over the benchmarks designed in Section 6.3 and presented in Tables 6.6, which are characterized by $o_p = \min$. First, in Section 8.2.1 we use the methodology introduced in Section 6.1 to evaluate the benchmarks by running UAQ-Solve leveraging Loandra. The result of the evaluation is that we demonstrate that the suite of benchmarks is complete and satisfactory. In particular, it consists of both easy and hard benchmarks which are empirically adequate. As already explained in Section 6.1, the methodology is also valid when the UAQ problem instances are encoded with joint optimization (with optimization priority p). Consequently, for the sake of completeness, we compare UAQ-Solve performance over the benchmarks Plb_bigR , R_bigPlb , and $RPhat_bigPlb$ with different optimization configurations. In particular:

1. $o_p = \min$ and $o_r = \text{any}$ (permission-based optimization);
2. $o_p = o_r = \min$;
3. $o_p = \min$ and $o_r = \max$;

In all the configurations, permission optimization is the priority ($pri = p$). The result of the experiment shows that joint optimization is not detrimental for performance. In fact, the median solving time of UAQ-Solve over the benchmarks encoded for joint optimization is similar to the one for permission-based optimization.

Finally, in Section 8.2.2 we experimentally evaluate the solvers 2D-Opt-Search and 2D-Opt-CNF by comparing their performance with UAQ-Solve performance. The reason why we decided to make this comparison is that the work by Mousavi et al. (64) is the most similar to our work. The results show that UAQ-Solve outperforms the other solvers over the vast majority of the instances.

8.2.1 Benchmarks evaluation

In the first step, we run UAQ-Solve over our benchmarks encoded with permission-based optimization, then with joint optimization.

Benchmarks parametric in $|P_{lb}|$

In Figures 8.4a and 8.4b we show respectively the median solving time and the percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|P_{lb}|$, namely Plb_bigR (pink) and Plb_smallR (blue). As expected, the former behaves like Algorithm 3 and exhibits a clear exponential growth with the increase in $|P_{lb}|$. In particular, UAQ-Solve reached the timeout for the 40% of the instances when $|P_{lb}| = 30$, while all the instances have been skipped when $|P_{lb}| = 40$. Also the median solving time over Plb_smallR meets the expectation by slightly varying between ~2 and ~6 seconds. In fact, the small value of $|R|$ would make it more convenient to use Algorithm 1, which is not affected by $|P_{lb}|$. None among the instances have been skipped.

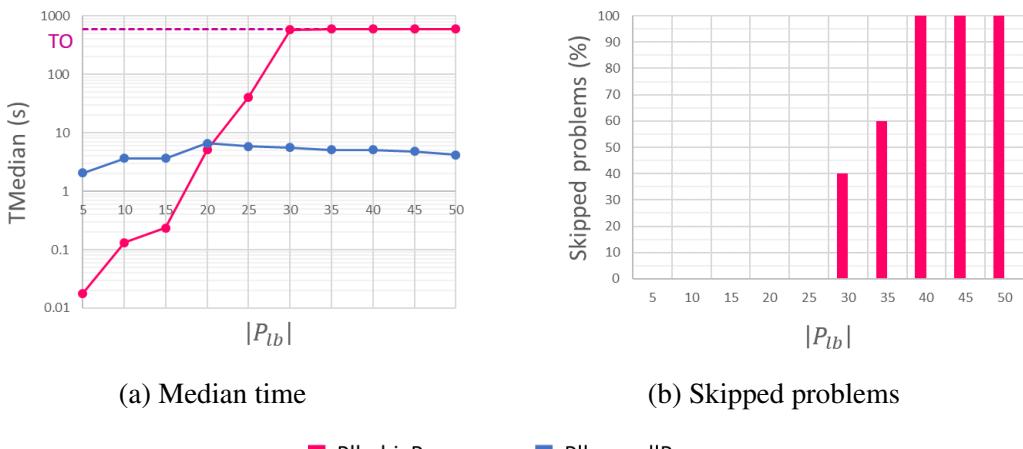


Figure 8.4 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|P_{lb}|$ with permission-based optimization

Benchmarks parametric in $|R|$

Figures 8.5a and 8.5b present respectively the median solving time and the percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|R|$, namely R_bigPlb (pink) and $R_smallPlb$ (blue). This experiment is dual to the previous one: the methodology foresees an exponential growth like Algorithm 1 over R_bigPlb ; on the contrary, due to the small value of $|P_{lb}|$, the methodology suggests that the algorithm that best performs over $R_smallPlb$ is Algorithm 3, whose complexity does not depend on the parameter under examination.

The results confirm our prediction: the median solving time of UAQ-Solve over R_bigPlb quickly grows from ~ 2 seconds when $|R| = 10$ to the timeout when $|R| = 40$. Note also that, from this value on, all the instances have been skipped, while UAQ-Solve could solve all the problems before the timeout when $|R| \leq 30$.

On the contrary, the median solving time even decreases over $R_smallPlb$. Besides, UAQ-Solve efficiently solved 100% of the instances.

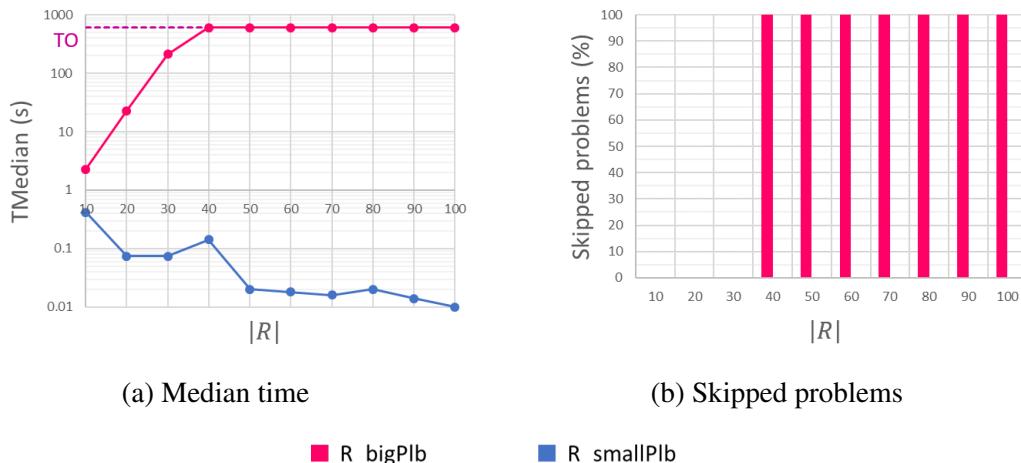


Figure 8.5 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|R|$ with permission-based optimization

Benchmarks parametric in \widehat{R}_P

Figure 8.6a presents the median solving time and the percentage of skipped problems of UAQ-Solve over the benchmarks parametric in \widehat{R}_P , namely $RPhat_bigPlb$ (pink), $RPhat_medPlb$ (blue) and $RPhat_smallPlb$ (yellow). This experiment is of particular interest: our methodology foresees that, with an adequately high number of roles, the complexity of this class of UAQ problem is polynomial with degree $|P_{lb}|$ as Algorithm 3 ($O(\widehat{R}_P^{|P_{lb}|})$). This is confirmed

by the results shown in Figure 8.6a: the latter shows the median solving time of UAQ-Solve over the three benchmarks, each with a different value for $|P_{lb}|$, i.e. 1 (*RPhat_smallPlb*), 4 (*RPhat_medPlb*) and 10 (*RPhat_bigPlb*). As expected, the complexity of the problem increases with the increasing of both \widehat{R}_P and $|P_{lb}|$: UAQ-Solve quickly solves all the problems in *RPhat_smallPlb*, while the plot is considerably more steep for *RPhat_bigPlb*. Besides, over *RPhat_bigPlb*, UAQ-Solve reached the timeout for 10% and 30% of the instances when \widehat{R}_P is 11 and 12 respectively. The results are consistent with the growth of polynomials of degree 1 and 10 respectively. The plot for *RPhat_mediumPlb* represents an intermediate situation.

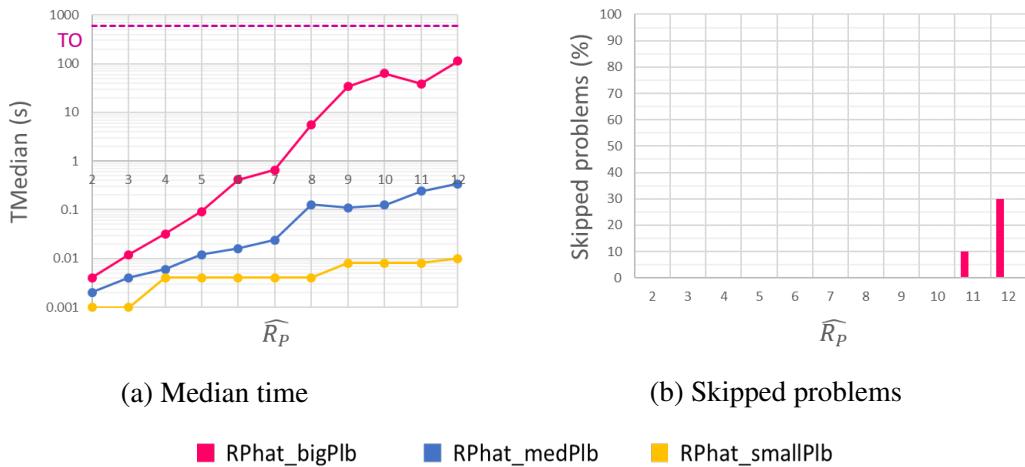


Figure 8.6 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in \widehat{R}_P with permission-based optimization

Other benchmarks

Figure 8.7 presents the median solving time of UAQ-Solve over the remaining benchmarks, namely *Pub*, *C*, *rshat*, and *that*, which are parametric in P_{ub} , $|C|$, \widehat{rs} and \widehat{t} respectively. We do not present the histograms representing the percentage of skipped problems because UAQ-Solve never reached the timeout over the aforementioned benchmarks.

The plot for *Pub* in Figure 8.7a indicates that the time spent by UAQ-Solve is not insensitive to $|P_{ub}|$. With the actual configuration used, Algorithm 3 is the best algorithm among the ones presented in Section 5.1 to solve the instances contained in the benchmark under examination. Consequently, the performance of a reasonable good solver over *Pub* should not be influenced by the increase in P_{ub} . However, in our benchmarks $P_{ub} = P$: we have to consider that, as presented in Section 5.2, the size of the encoding is in $O(|R||P|)$

and thus it grows linearly with $|P|$. Therefore, a slight increase in the median solving time over Pub is not surprising: it could be caused more by the nature of the encoding then by the difficulty of the generated problems itself.

Figures 8.7b, 8.7c, and 8.7d show the results of the experiments over the benchmarks parametric in the features of the MER constraints. Among the algorithms presented in Section 5.1, only Algorithm 4 is sensitive to the variation of these parameters. However, this algorithm is not adequate to solve UAQ problems with $o_p = \min$. Consequently, any reasonably performing solver should not be affected by the variation of the MER constraint features. The results of UAQ-Solve over C and $that$ confirm the expectation, with a median solving time around 0.1 seconds and 0.01 second respectively over all the instances considered. Instead, the median solving time moderately grows with $\hat{r}s$, however the increase (0.56 second) is not relevant, given the large range of the parameter considered ($5 \leq \hat{r}s \leq 50$).

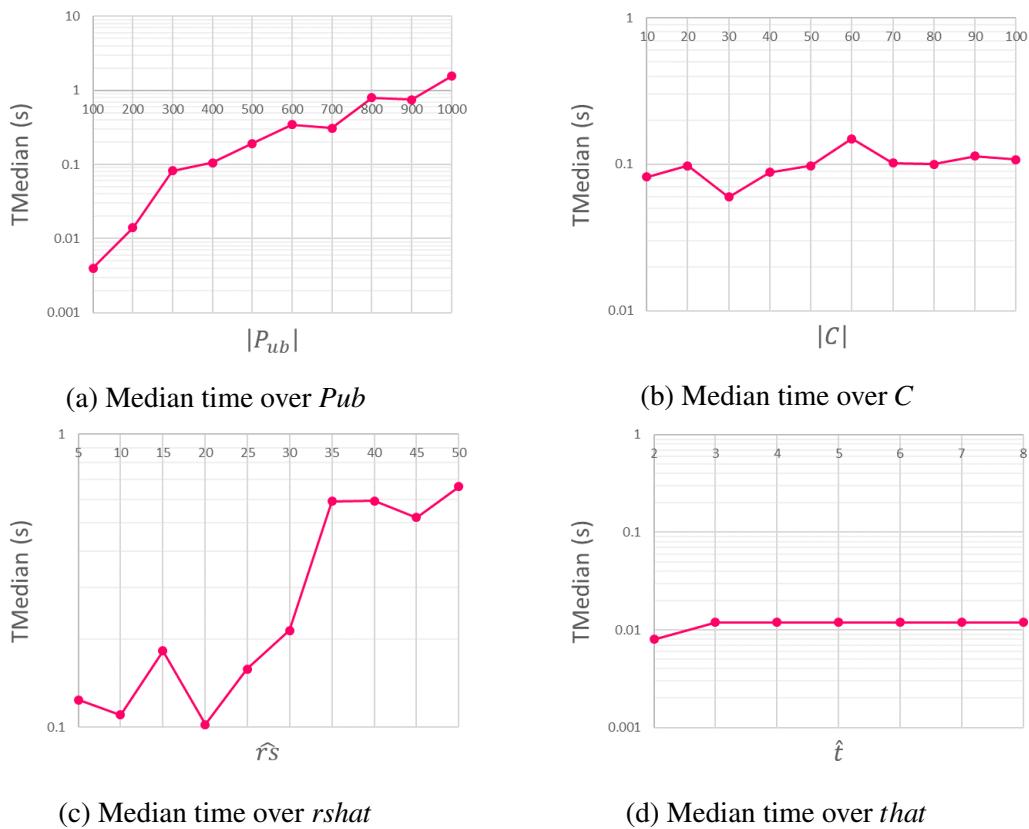


Figure 8.7 Median solving time of UAQ-Solve over the benchmarks parametric in $|P_{ub}|$, $|C|$, $\hat{r}s$ and \hat{t} with permission-based optimization

Permission-based optimization versus joint optimization

Figure 8.8 shows the results of the execution of UAQ-Solve over Plb_bigR , R_bigPlb , and $RPhat_bigPlb$ under different optimization configuration: permission-based optimization (pink), joint optimization with $o_p = o_r = \min$ (blue) and joint optimization with $o_p = \min$ and $o_r = \max$ (yellow).

For each benchmark considered, the shape of the three plots is similar, as expected. In particular, the curves related to permission-based optimization and joint optimization with $o_r = \min$ almost overlap, while a decrease in performance occurs when $o_r = \max$. This is evident also in the figures representing the percentage of skipped problems: the pink and the blue histograms are similar for Plb_bigR , while they coincide for R_bigPlb . Over $RPhat_bigPlb$ performance in case of joint optimization with $o_r = \min$ is even better than the case with simple permission-based optimization. In fact, in the former case UAQ-Solve never reached the timeout, while in case of permission-based optimization the percentage of skipped problems is 10% when $\widehat{R}_P = 11$ and 30% when $\widehat{R}_P = 12$. Instead, in case of $o_r = \max$, the percentage of skipped problems is always higher than in the other two cases: this is particularly evident for $RPhat_bigPlb$ over which the solver reached the timeout for the 60% of the instances when $\widehat{R}_P \geq 10$.

The fact that it is easier for a PMaxSAT solver to deal with problems with $o_p = o_r = \min$ is rather intuitive given the correlation among the activation of roles and the activation of permissions in the encoding⁶, see Section 5.2.

⁶ $(\neg \bar{r} \vee \bar{p})$ for all $p \in P$ and $r \in R_p$ and $(\neg \bar{p} \vee \bigvee \{\bar{r} : r \in R_p\})$ for all $p \in P$.

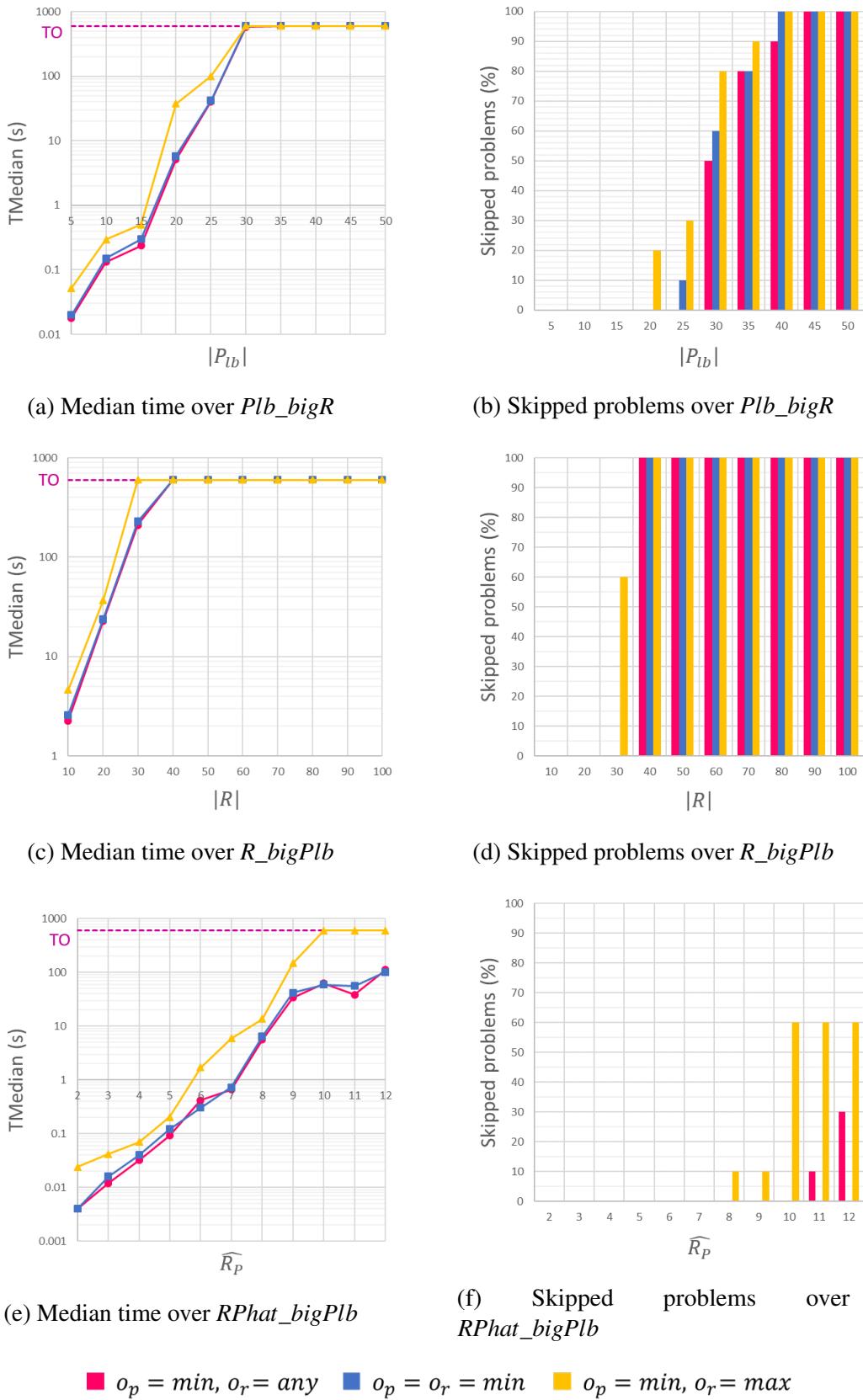


Figure 8.8 Median solving time and percentage of skipped problems of UAQ-Solve over *Plb_bigR*, *R_bigPlb* and *RPhat_bigPlb* with different optimization configurations

8.2.2 Solvers evaluation

In the previous subsection, we evaluated the benchmarks proposed and presented in Table 6.6. All of them satisfy the expectations and can thus be used to evaluate different solvers' performance. In particular, the benchmarks of hard instances, namely Plb_bigR and R_bigPlb , can be used to stress-test the solvers, while the other ones can be used to check the efficiency of the solvers. In particular, in this section we compare UAQ-Solve with solvers that tackle the UAQ problem through a different approach, namely 2D-Opt-Search and 2D-Opt-CNF.

Benchmarks parametric in $|P_{lb}|$

Figure 8.9 shows the results of the experiment over the benchmarks parametric in $|P_{lb}|$, namely Plb_bigR (hard benchmark, above) and Plb_smallR (easy benchmark, below).

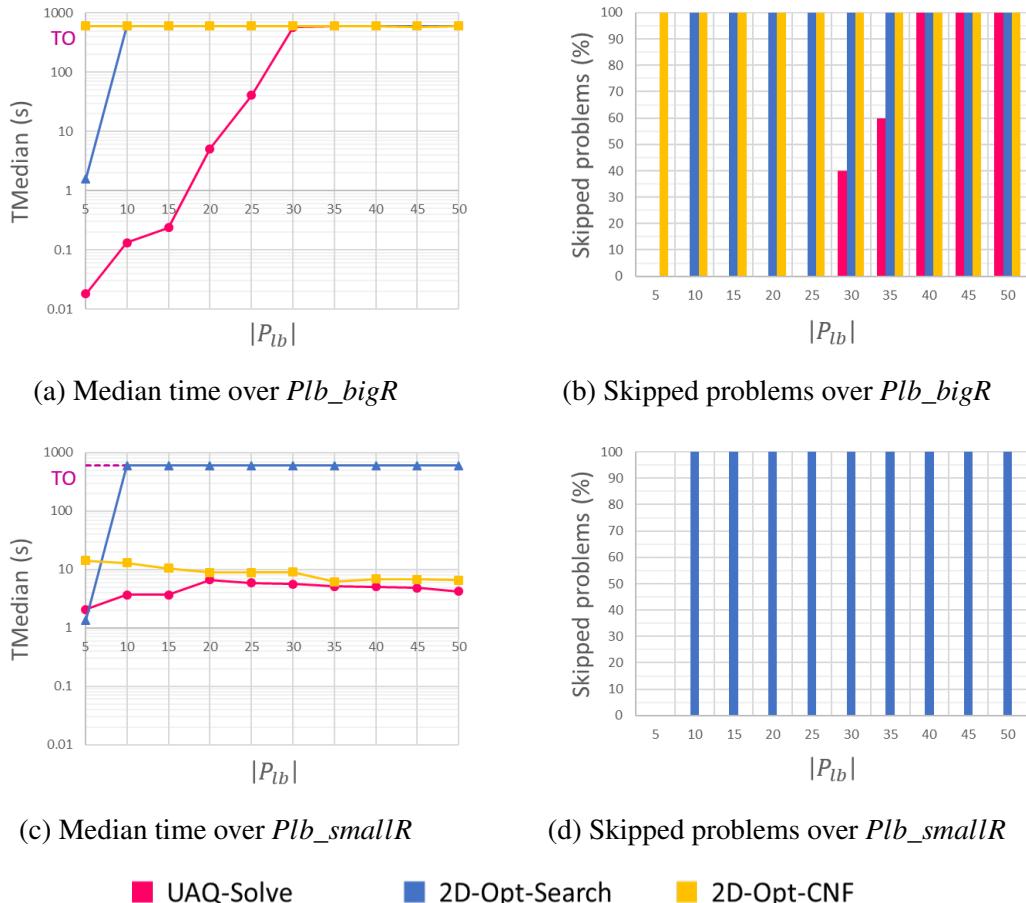


Figure 8.9 Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks parametric in $|P_{lb}|$ with permission-based optimization

In particular, the plots in Figure 8.9a and the histograms in Figure 8.9b respectively represent the median solving time and the percentage of skipped problems of the three solvers over Plb_bigR . It is evident that UAQ-Solve outperforms both 2D-Opt-Search and 2D-Opt-CNF. The median solving time of 2D-Opt-Search quickly increases from 1.58 seconds when $|P_{lb}| = 5$ to the timeout when $|P_{lb}| = 10$. Besides, when $|P_{lb}| \geq 10$, the percentage of skipped problems is always 100%. 2D-Opt-CNF perform even worst: in fact, all the instances have been skipped even for the smallest value of $|P_{lb}|$.

As presented in Figures 8.9c and 8.9d, UAQ-Solve outperforms 2D-Opt-Search and 2D-Opt-CNF also over Plb_smallR . However, in this case, 2D-Opt-CNF performs more similarly to UAQ-Solve, which efficiently solves all the instances. On the contrary, 2D-Opt-Search reaches the timeout for all the instances when $|P_{lb}| \geq 10$.

Benchmarks parametric in $|R|$

Figure 8.10 shows the median solving time and the percentage of skipped problems of the three solvers under examination over R_bigPlb (hard benchmark, above in the figure) and $R_smallPlb$ (easy benchmark, below in the figure). UAQ-Solve outperforms the other solvers over the difficult benchmark. 2D-Opt-CNF performs similarly to UAQ-Solve over R_bigPlb , even if it reaches the timeout for the majority of the instances already when $|R| = 20$. The percentage of skipped problems increases to 80% when $|R| = 30$ and then to 100% when $|R| = 40$. Instead, 2D-Opt-Search reached the timeout for the 100% of the instances even for the smallest value of the parameter.

Notice that 2D-Opt-Search performs remarkably well for $R_smallPlb$: in fact, it even outperforms UAQ-Solve when $|R| \leq 40$. On the contrary, 2D-Opt-CNF quickly reaches the timeout also over the easy benchmark: in particular, the percentage of skipped problems is 100% when $|R| \geq 20$. It seems then evident that 2D-Opt-CNF in general does not scale well with the increasing of the number of roles.

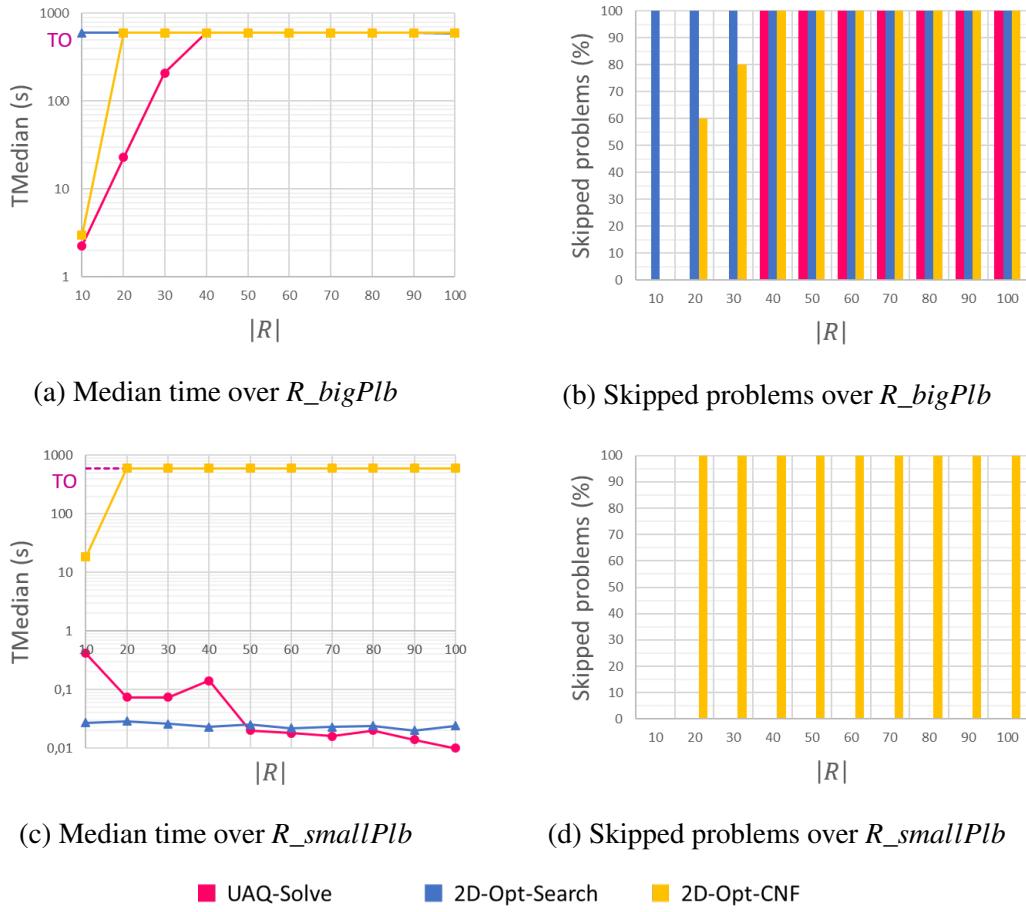


Figure 8.10 Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks parametric in $|R|$ with permission-based optimization

Benchmarks parametric in \widehat{R}_P

Figure 8.11 presents the median solving time and the percentage of skipped problems of UAQ-Solve, 2D-Opt-Search and 2D-Opt-CNF over the benchmarks parametric in \widehat{R}_P , namely RP_bigPlb (above in the figure), RP_medPlb (in the middle) and $RP_smallPlb$ (below in the figure), which are characterized by $|P_{lb}| = 10$, $|P_{lb}| = 4$ and $|P_{lb}| = 1$ respectively⁷.

⁷We remember that a reasonably good solver should behave like a polynomial of degree $|P_{lb}|$.

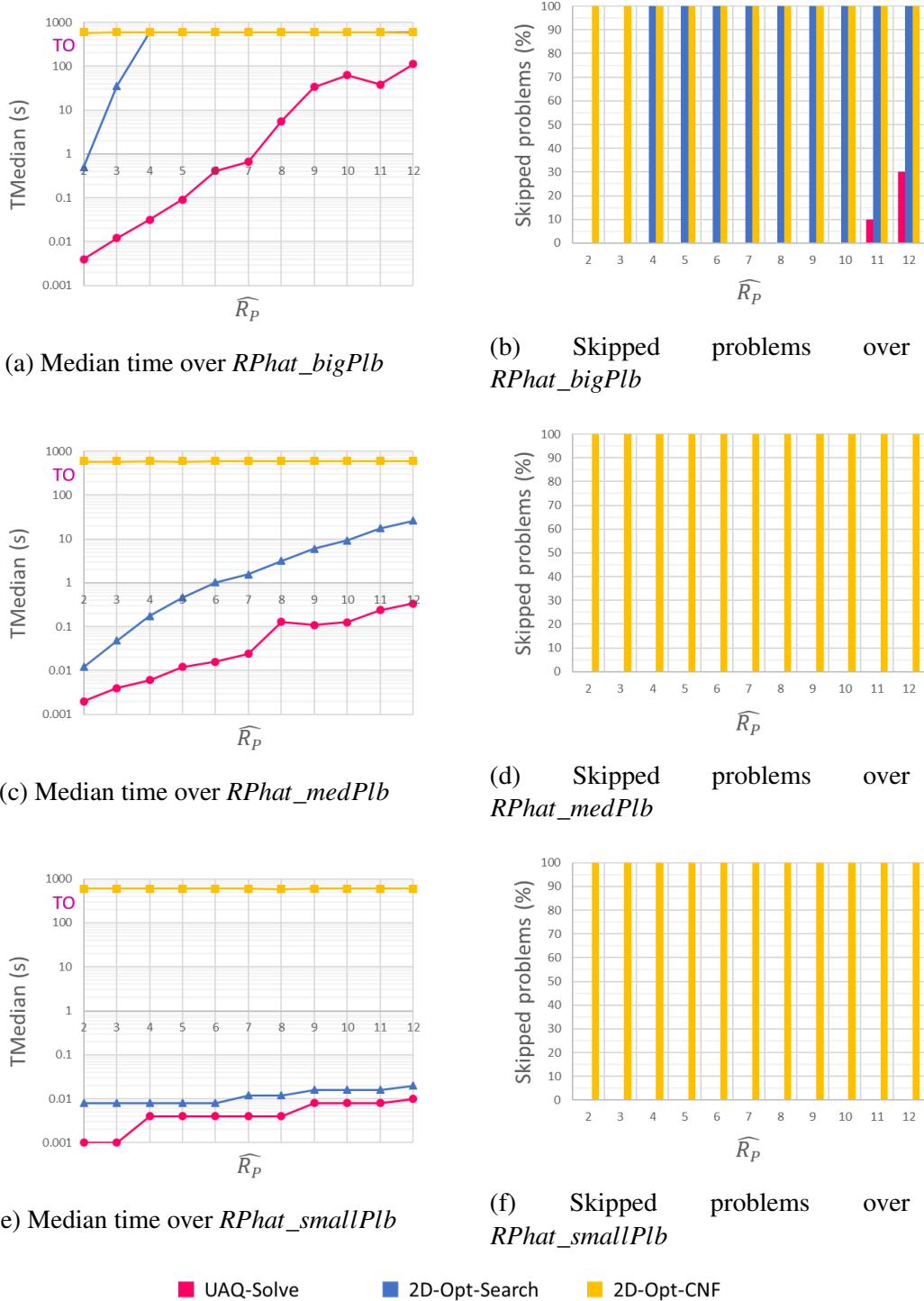


Figure 8.11 Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmarks parametric in \widehat{R}_P with permission-based optimization

2D-Opt-Search behaves like than UAQ-Solve, even if less performing: in fact, the difference in the slopes due to the value of $|P_{lb}|$ is evident. However, 2D-Opt-Search over *RPhat_bigPlb* reaches the timeout for all the instances when $\widehat{R}_P \geq 4$, while the percentage of problems skipped by UAQ-Solve is 10% when $\widehat{R}_P = 11$ and 30% when $\widehat{R}_P = 12$.

On the contrary, it is clear that 2D-Opt-CNF does not perform well on this class of UAQ problem: in fact, it always reaches the timeout for 100% of the instances contained in the benchmarks considered, regardless of the value of \widehat{R}_P .

Benchmark parametric in $|P_{ub}|$

Figure 8.12 shows the results of the experiment over the benchmark parametric in $|P_{ub}|$, namely *Pub*. The latter is supposed to be an easy benchmark: in fact, the median solving time of UAQ-Solve over *Pub* increases with $|P_{ub}|$, however it never reaches the timeout.

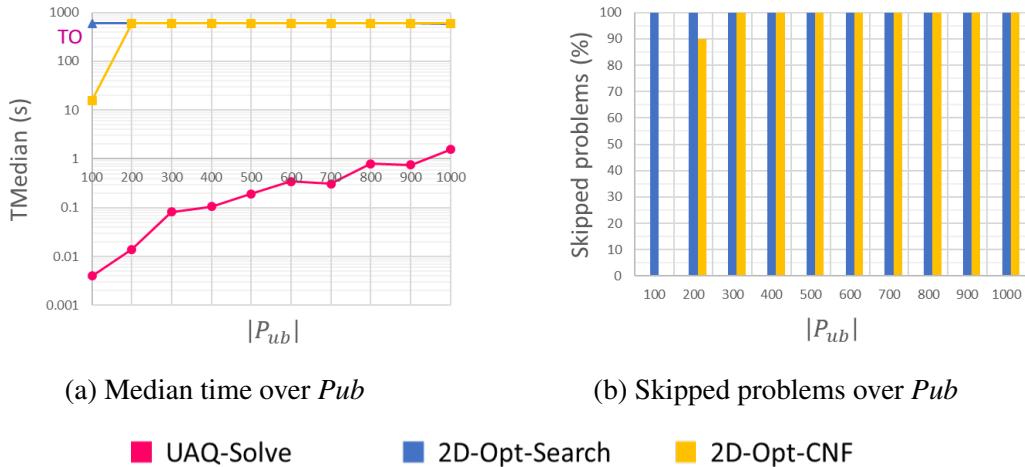


Figure 8.12 Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in $|P_{ub}|$ with permission-based optimization

Nevertheless, both 2D-Opt-Search and 2D-Opt-CNF do not perform well over the benchmark. In particular, the former reaches the timeout for all the instances included in the benchmark. Instead, the latter solves all the instances with $|P_{ub}| = 100$ in ~ 16 seconds (compared to the 4 milliseconds achieved by UAQ-Solve); then the 90% of the instances have been skipped when $|P_{ub}| = 200$; finally, 2D-Opt-CNF skipped all the remaining instances.

We already noticed in Section 8.2.1 that our approach is a little sensitive to the variation of $|P_{ub}|$ due to the encoding; however, 2D-Opt-Search and 2D-Opt-CNF perform far worse.

Benchmark parametric in $|C|$, $\hat{r}s$, and \hat{t}

This section presents the results of the execution of UAQ-Solve, 2D-Opt-Search, and 2D-Opt-CNF over the benchmarks parametric in $|C|$, $\hat{r}s$, and \hat{t} . As already demonstrated in Section 8.2.1, C , $rshat$, and $that$ are easy benchmarks: in fact, the complexity of the UAQ problem depends on the MER constraints features only when $o_p = \max$.

However, as can be seen in Figures 8.13 and 8.14, while UAQ-Solve solves all the instances contained in C and $rshat$ in a fraction on a second, both 2D-Opt-Search and 2D-Opt-CNF could never find the optimum solution before the timeout: the percentage of skipped problems of the two solvers over C and $rshat$ is 100% for all the values of the two parameters for which we have evidence.

On the contrary, as depicted in Figure 8.15, 2D-Opt-Search performs similarly to UAQ-Solve over $that$, by solving all the instances in 20 milliseconds. By the way, 2D-Opt-CNF still is ineffective over $that$: in fact, it takes 132 seconds to solve the problems for the smallest value of \hat{t} (i.e. 2), then it reaches the timeout for the 90% of the instances when $\hat{t} = 3$ and, after that value of the parameter, for the 100% of the instances.

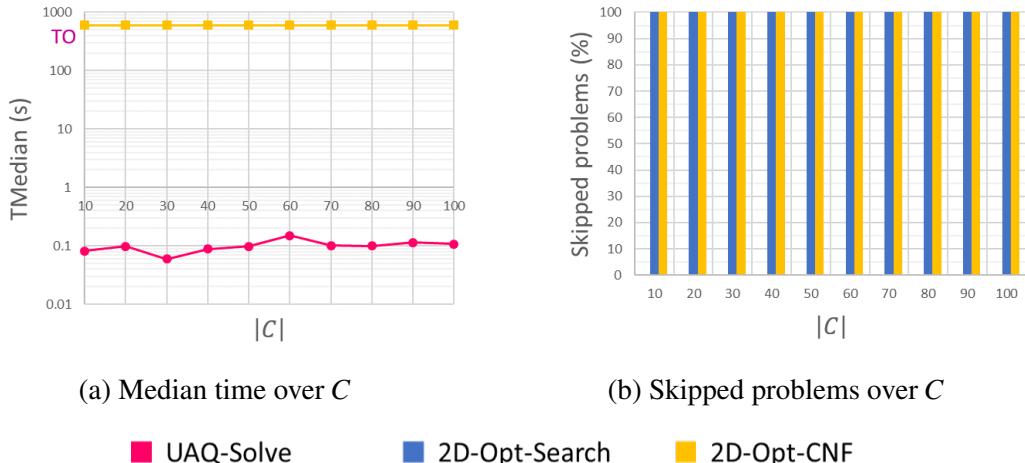


Figure 8.13 Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in $|C|$ with permission-based optimization

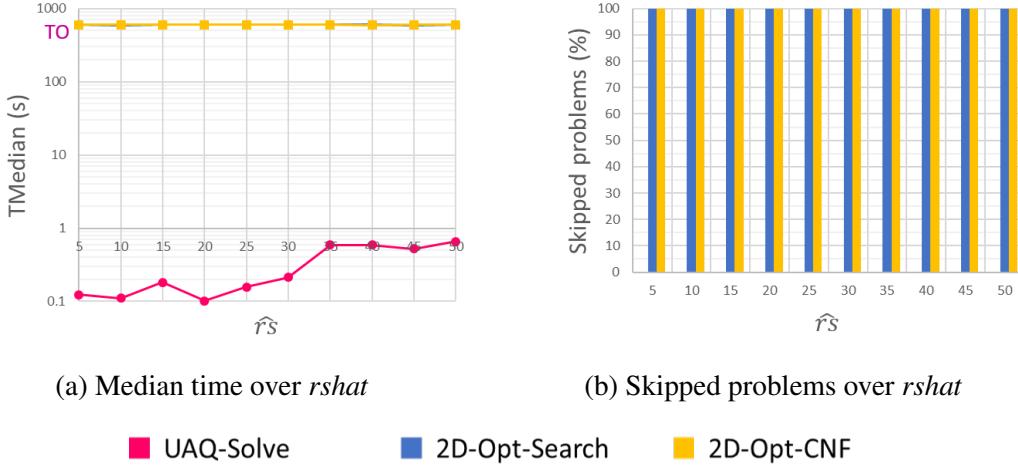


Figure 8.14 Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in $\hat{r}s$ with permission-based optimization

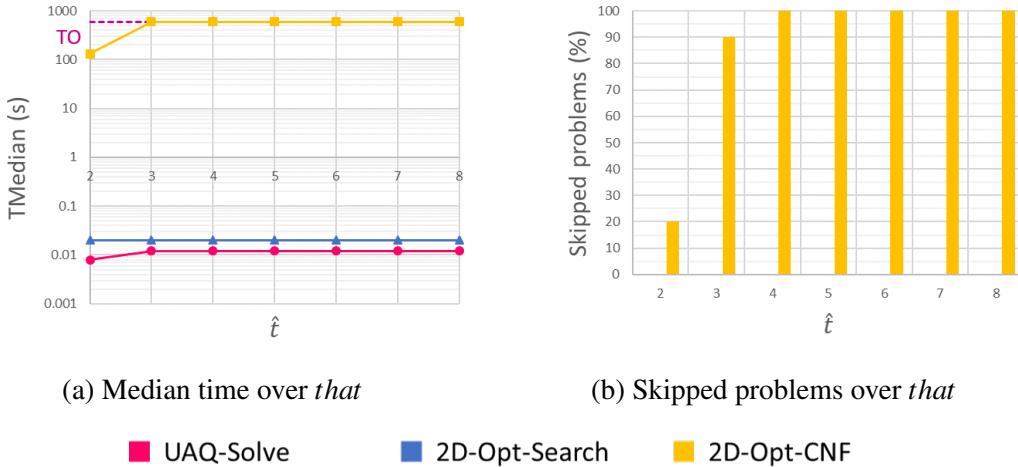


Figure 8.15 Median solving time and percentage of skipped problems of 2D-Opt-Search, 2D-Opt-CNF and UAQ-Solve over the benchmark parametric in \hat{t} with permission-based optimization

8.3 Our benchmarks with $o_p = \max$

This section presents the evaluation of the benchmarks designed in Section 6.3 and presented in Table 6.7, which are characterized by $o_p = \max$. In the same way as we did in Section 8.3 for the benchmarks designed for $o_p = \min$, we use the methodology introduced in Section 6.1

and run UAQ-Solve leveraging Loandra over the benchmarks. As a result of the evaluation, we can state that we provide an empirically adequate suite of benchmarks that includes both easy and hard benchmarks. UAQ-Solve behaves as predicted by the methodology over all the benchmarks but the one parametric $|P_{ub}|$. In fact, contrary to our expectations, UAQ-Solve over this benchmark exhibits a clear exponential growth. More investigations on this behavior are needed.

For the sake of completeness, we also compare UAQ-Solve performance over the benchmarks R_bigCt , Pub , $that_bigRPub$, C_bigR , and $rshat_bigCt$ with different optimization configurations. In particular:

1. $o_p = max$ and $o_r = any$ (permission-based optimization);
2. $o_p = o_r = max$;
3. $o_p = max$ and $o_r = min$;

In all the configurations, permission optimization is the priority ($pri = p$). The results of the comparison show that performance degrades with joint optimization.

8.3.1 Benchmarks evaluation

In the first step, we run UAQ-Solve over our benchmarks encoded with permission-based optimization, namely with $o_p = max$ and $o_r = any$.

Benchmarks parametric in $|R|$

Figure 8.16 depicts the median solving time and the percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|R|$, namely R_bigCt (pink) and $R_smallCt$ (blue). As the name itself suggests, the former is characterized by high values for C and \hat{t} , which make Algorithm 1 the more adequate algorithm among the ones discussed to use to solve the problems included in the benchmarks. Since Algorithm 1 is exponential with respect to $|R|$, we expect the same behavior when running UAQ-Solve over the benchmarks. This expectation is verified by the experiment: the median solving time clearly grows exponentially with the increase of $|R|$. UAQ-Solve can solve all the instances before the timeout when $|R| \leq 60$; after that value, the percentage of skipped problems is always 100%.

On the contrary, since the value of $|C|\hat{t}$ is always higher than $|R|$, Algorithm 4 is the most suitable over $R_smallCt$. Since the algorithm's complexity does not depend on the parameter under examination, we do not expect a substantial growth of the median solving time. Also

in this case, our expectation is confirmed: UAQ-Solve could solve all the instances proposed at most in 8 milliseconds.

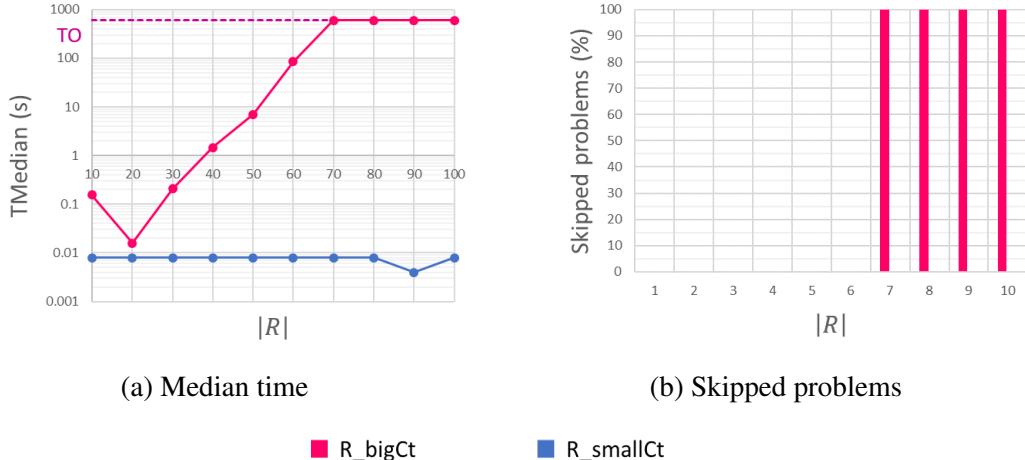


Figure 8.16 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|R|$ with permission-based optimization

Benchmarks parametric in $|P_{ub}|$

Figure 8.17 presents the median solving time and the percentage of skipped problems of UAQ-Solve over P_{ub} . The time spent by UAQ-Solve to solve P_{ub} does not meet the expectations: in fact the plot has a clear exponential growth. This behavior can be due to the fact that this class of problems is difficult to tackle for the PMaxSAT solver; another possibility is that, by chance, we generated particularly difficult instances. This result needs to be better addressed in the future.

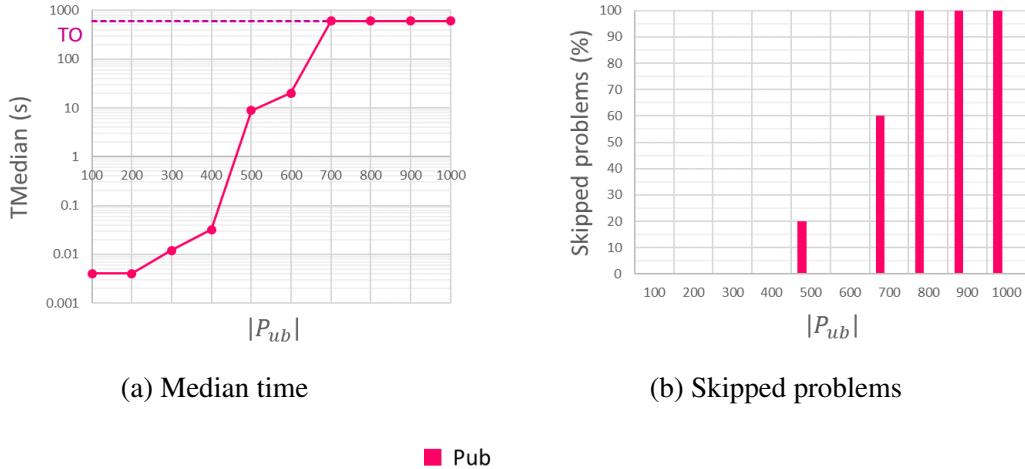


Figure 8.17 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmark parametric in $|P_{ub}|$ with permission-based optimization

Benchmarks parametric in $|C|$

Figure 8.18 presents the median solving time and the percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|C|$, namely C_bigR and C_smallR .

Our methodology foresees that a reasonably good solver should solve C_bigR in exponential time like Algorithm 4, while over C_smallR it should behave like Algorithm 1, whose complexity is not influenced by the parameter under examination.

The results satisfy both the expectations. UAQ-Solve exhibits a clear exponential behavior over C_bigR : in particular, it reaches the timeout for the 30% of the instances when $|C| = 60$, for half the instances when $|C| = 70$, and later for all the instances. On the contrary, UAQ-Solve solves all the instances contained in C_smallR in around 10 milliseconds.

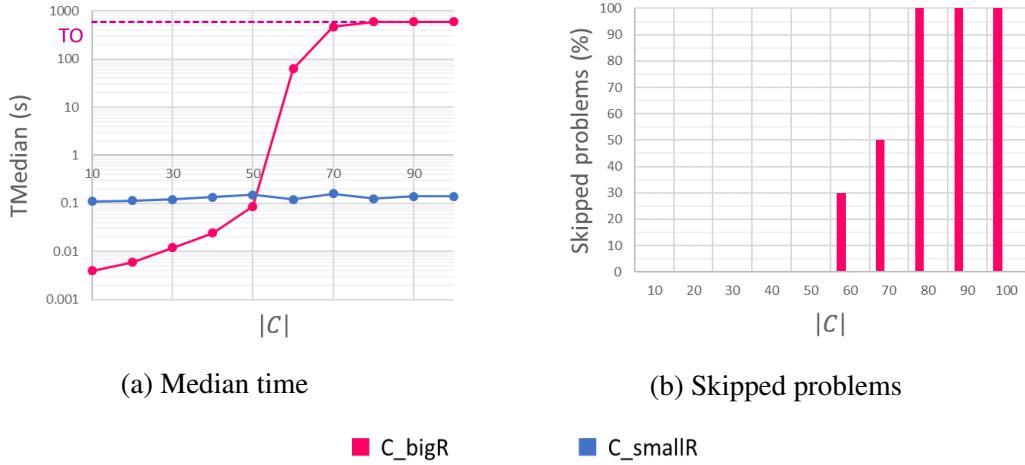


Figure 8.18 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $|C|$ with permission-based optimization

Benchmarks parametric in \hat{t}

Figure 8.19 presents the median solving time and the percentage of skipped problems of UAQ-Solve over the benchmarks parametric in \hat{t} , namely *that_bigR* (pink) and *that_smallR* (blue). According to our methodology, any solver should behave like Algorithm 3 and consequently exhibit an exponential behavior with the growth in \hat{t} . On the contrary, any reasonable good solver over *that_smallR* should be insensitive to the variation of \hat{t} , like Algorithm 1.

As expected, the median solving time of UAQ-Solve over *that_bigR* grows exponentially. Notice that when $\hat{t} = 2$, even if the median solving time is 80 milliseconds, the percentage of skipped problems is already 20%. Then, it rapidly increases to 90% (when $\hat{t} = 3$) and finally to 100% (when $\hat{t} \geq 4$).

On the contrary, UAQ-Solve quickly solves the instances contained in *that_smallR*, even if the median solving time is 238 milliseconds for the smallest value of the parameter under examination and then decreases to 4 milliseconds.

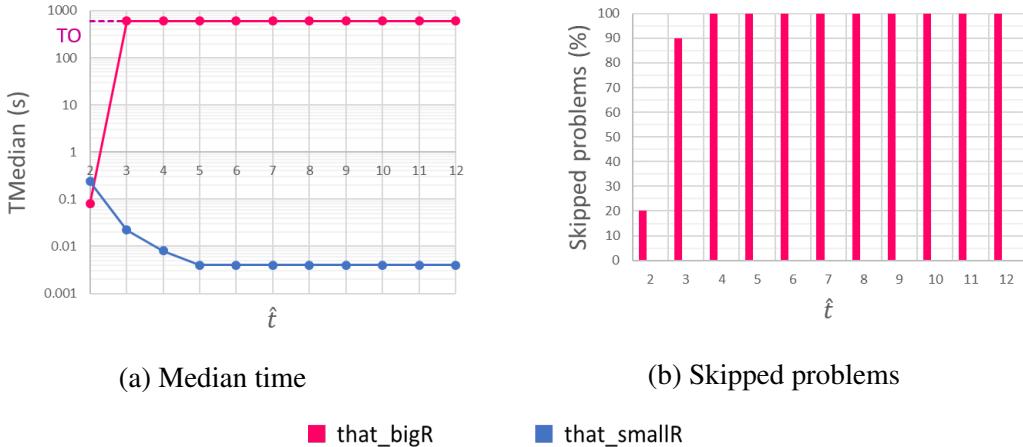


Figure 8.19 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in \hat{t} with permission-based optimization

Benchmarks parametric in \hat{rs}

Figure 8.20 presents the median solving time and the percentage of skipped problems of UAQ-Solve over the benchmarks parametric in \hat{rs} , namely *rshat_bigCt* (pink), *rshat_medCt* (blue) and *rshat_smallCt* (yellow). The latter are characterized by $|C|\hat{t} = 30$, $|C|\hat{t} = 9$ and $|C|\hat{t} = 3$ respectively. The benchmarks were designed adequately so that any reasonably good solver performs over them like Algorithm 3, by exhibiting a polynomial behavior with degree $|C|\hat{t}$ with the increase of \hat{rs} .

This experiment is of particular interest; in fact, it demonstrates the influence of different values of $|C|\hat{t}$: the plots show that the problems can be solved quickly as \hat{rs} increases as long as the value of $|C|\hat{t}$ is sufficiently small, like 3 for *rshat_smallCt*. The plots for *rshat_medCt*, and *rshat_bigCt* show the effect of larger values of $|C|\hat{t}$, which is consistent with the growth of polynomials of degree 9 and 30 respectively. In particular, in the latter case, the percentage of skipped problems is 90% when $\hat{rs} = 25$ and 100% when $\hat{rs} = 30$.

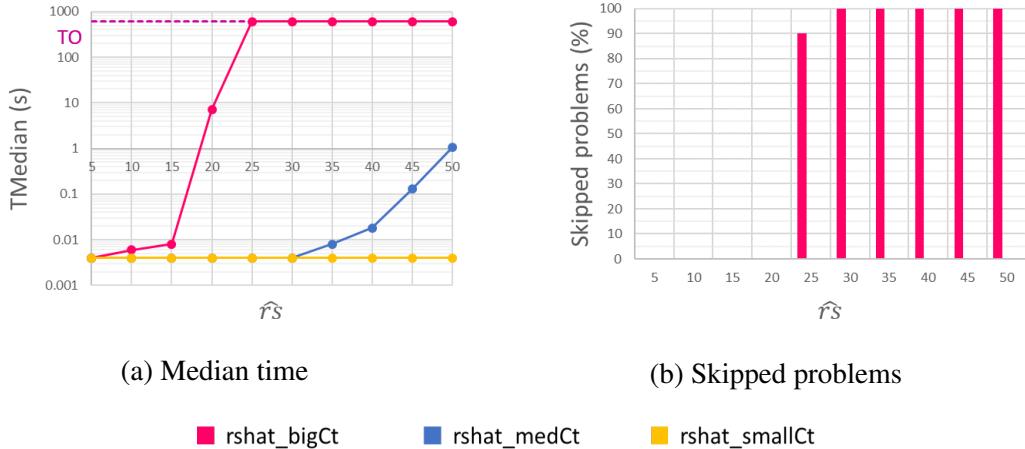


Figure 8.20 Median solving time and percentage of skipped problems of UAQ-Solve over the benchmarks parametric in $\hat{r}s$ with permission-based optimization

Other benchmarks

Figures 8.21a and 8.21 show the median solving time of UAQ-Solve respectively over $R\widehat{P}$, which is parametric in \widehat{R}_P , and Plb , which is parametric in $|Plb|$. Given the values used for the UAQ problem dimensions during the design phase, the methodology foresees that a reasonably good solver performs similarly to Algorithm 1 over $R\widehat{P}$ and Algorithm 3 over Plb . The complexity of the two algorithms does not depend on the parameters under consideration; consequently, we expect that UAQ-Solve quickly solves all the instances in the benchmarks. The plots in Figure 8.21 confirm our expectations. We do not present the histograms representing the percentage of skipped problems, because UAQ-Solve did not skip any instance contained in the two benchmarks.

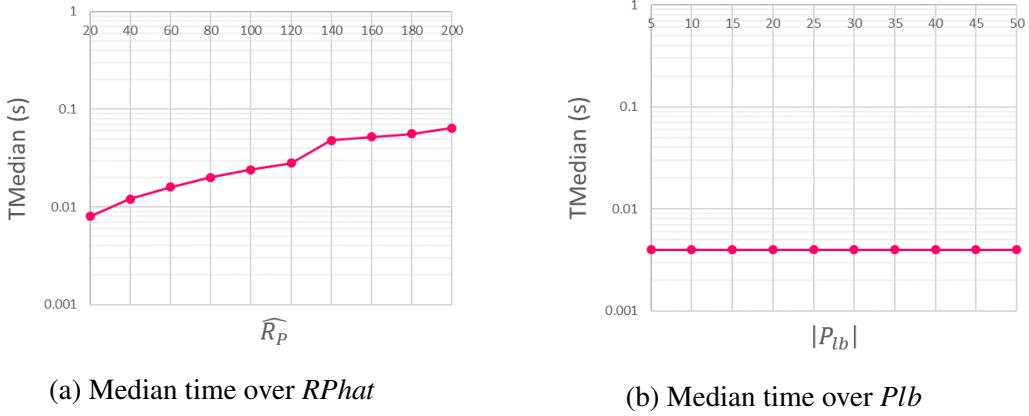


Figure 8.21 Median solving time of UAQ-Solve over the benchmarks parametric in \widehat{R}_P and $|P_{lb}|$ with permission-based optimization

Permission-based optimization versus joint optimization

In this section, we compare the performance of UAQ-Solve over *R_bigCt*, *Pub*, *that_bigR*, *C_bigR*, and *rshat_bigCt* encoded with different optimization objectives. In particular, for each benchmark, we show the median solving time and the percentage of skipped problems with the following encodings: permission-based optimization (pink), joint optimization with maximization of both permissions and roles as objectives (blue) and joint optimization with maximization of permissions and minimization of roles as objectives (yellow).

Figure 8.22 shows the results of the execution of UAQ-Solve over *R_bigCt*. It is evident that in this case joint optimization minimally affects performance: in fact, the curves representing the median solving time are almost overlapping. There is only a little difference in the percentage of skipped problems: in fact, when, $|R| = 60$, it is 0% in case of permission-based optimization and 10% in the other cases.

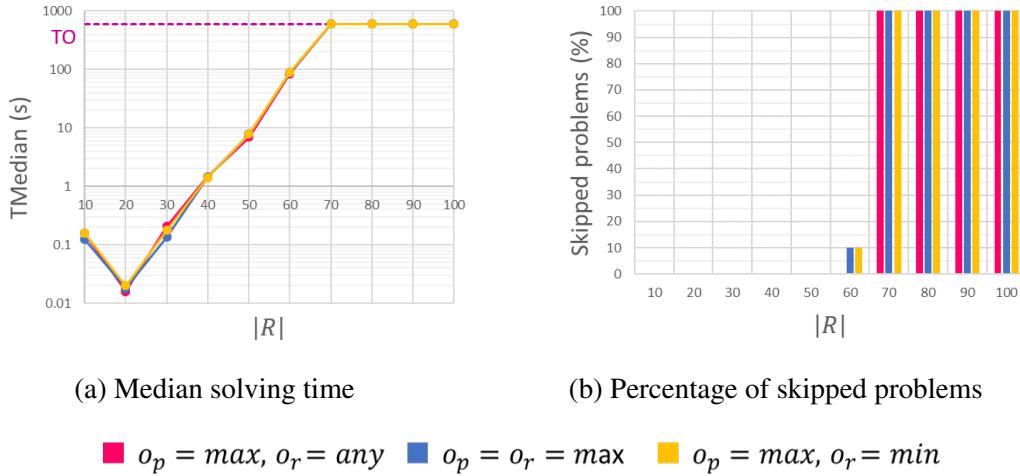


Figure 8.22 Median solving time and percentage of skipped problems of UAQ-Solve over R_bigCt with different optimization configurations

Differently from the previous case, joint optimization highly influence UAQ-Solve performance over Pub . As shown in Figure 8.23a, while in case of permission-based optimization the median solving time exponentially increases until it reaches the timeout with $|P_{lb}| = 700$, in the other two cases the percentage of skipped problem is 100% for each value of the parameter under examination.

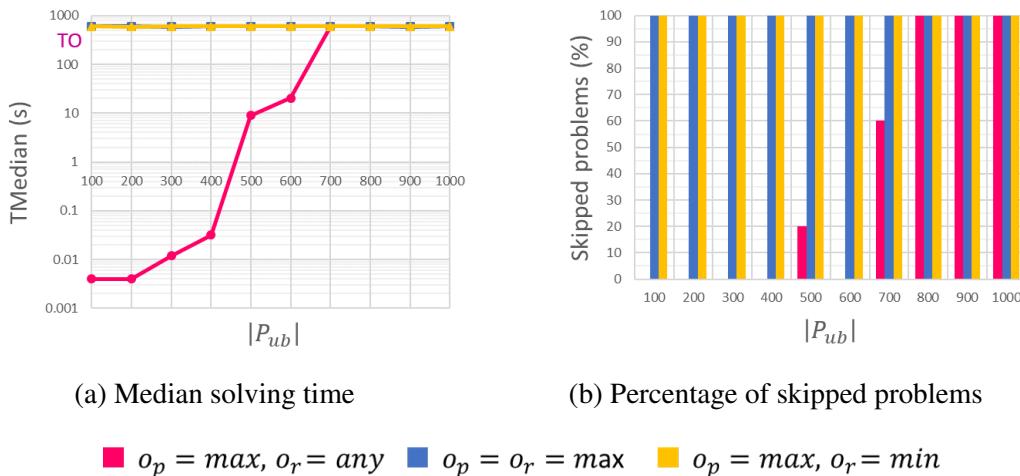


Figure 8.23 Median solving time and percentage of skipped problems of UAQ-Solve over Pub with different optimization configurations

With regard to *that_bigRPub*, in case of permission-based optimization, the median solving time of UAQ-Solve over it is different from the timeout only for the smallest value

of the parameter, namely for $\hat{t} = 2$. As shown in Figure 8.24, performance further degrades with joint optimization: when the role objective is different from *any*, the solver immediately reaches the timeout for most of the instances.

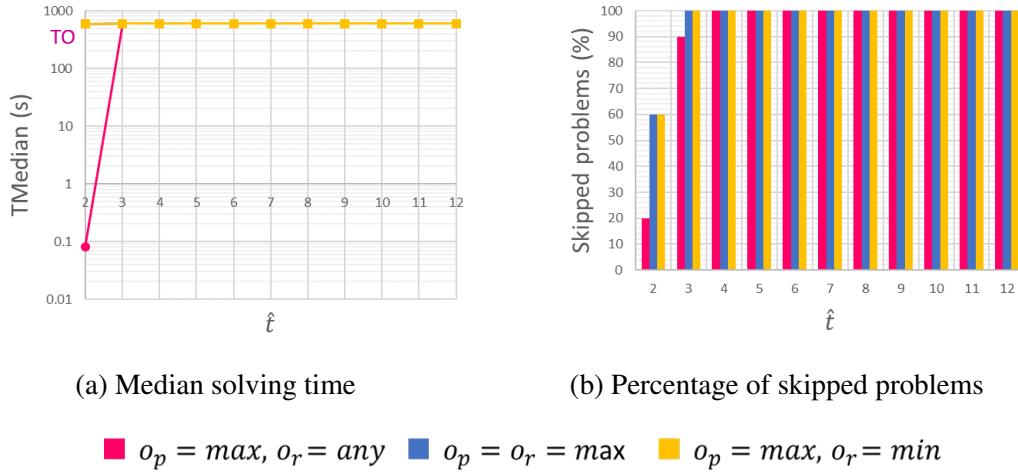


Figure 8.24 Median solving time and percentage of skipped problems of UAQ-Solve over *that_bigRPub* with different optimization configurations

Figures 8.25 and 8.26 present the results of the execution of UAQ-Solve over *C_bigR* and *rshat_bigCt* respectively. Over both the benchmarks, a performance degradation occurs in case of joint optimization. However, over both the benchmarks, the shape of the median solving time in case of $o_r = max$ is similar to the one related to permission-based optimization. On the contrary, when $o_r = min$, UAQ-Solve over the two benchmarks can not solve almost the totality of the instances before the timeout.

The result is consistent with the observation we made in Section 8.2.1, namely that, in case of joint optimization, it is easier for a PMaxSAT solver to solve problems in which $o_r = o_p$.

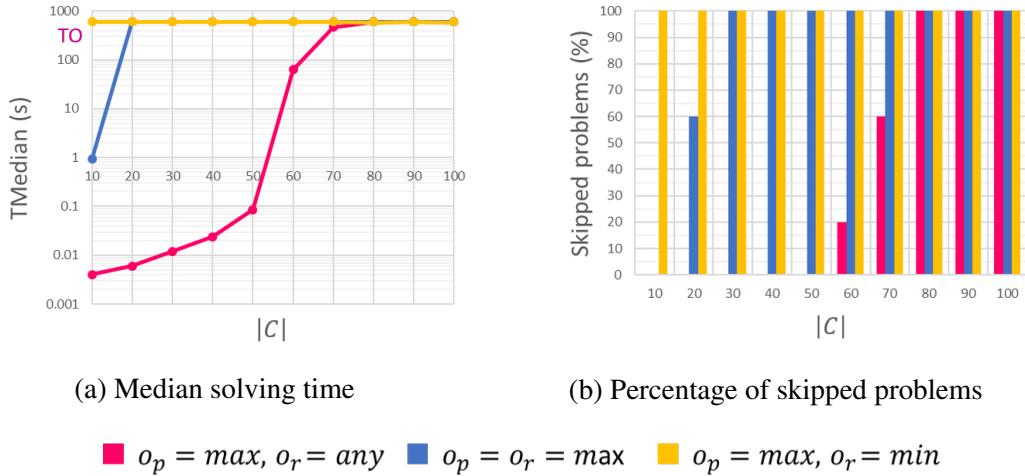


Figure 8.25 Median solving time and percentage of skipped problems of UAQ-Solve over C_bigR with different optimization configurations

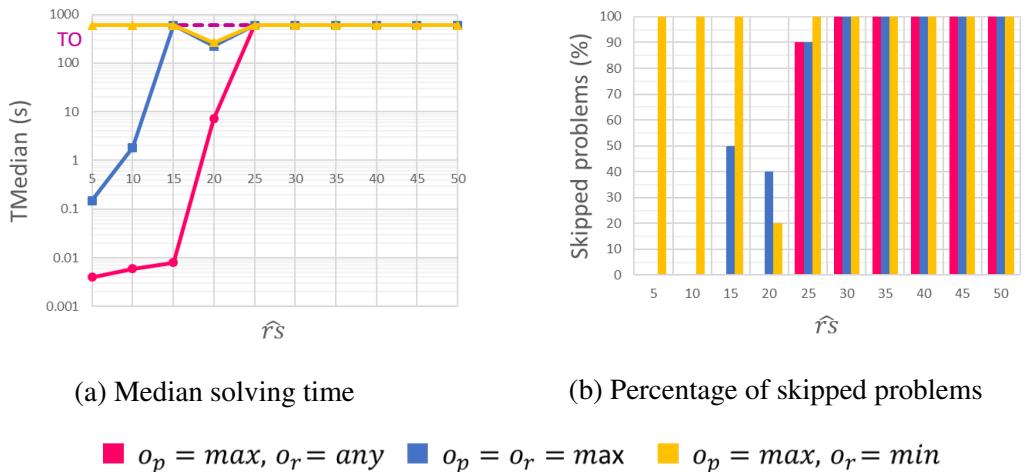


Figure 8.26 Median solving time and percentage of skipped problems of UAQ-Solve over $rshat_bigCt$ with different optimization configurations

8.4 Solvers evaluation through our benchmarks

In the previous sections, we used the methodology introduced in Section 6.1 to evaluate

- the benchmarks from (64) and presented in Table 6.4 (Section 8.1),
- the benchmarks presented in Table 6.6, which we designed and generated leveraging the methodology summarized in Table 6.2 (Section 8.2), and

- the benchmarks presented in Table 6.7, which we designed and generated leveraging the methodology summarized in Table 6.3 (Section 8.3).

The evaluation proved that the benchmarks from (64) are incomplete and unsatisfactory. On the contrary, the benchmarks designed through our methodology are empirically adequate. In particular, we provide a set of benchmarks that includes both easy and hard problems. The first ones can be used to evaluate the solvers' performance over easy problems, while the second ones allow to stress-test the solvers. We can then use the benchmarks to compare the performance of different solvers over the different classes of UAQ problem instances. In particular, in this section we compare the performance of different PMaxSAT solvers over Plb_bigR , R_bigPlb and $RPhat_bigPlb$ with $o_p = min$, $o_r = any$ and $pri = p$ (permission-based optimization) and Pub , C_bigR , $rshat_bigCt$ and $that_bigRPub$ with $o_p = max$, $o_r = any$ and $pri = p$ (permission-based optimization). To this aim, UAQ-Solve makes use of the following PMaxSAT solvers:

- LMHS;
- Loandra;
- MaxHS;
- Maxino;
- Open-WBO-RES;
- QMaxSAT;
- QMaxSATuc;
- WPM1-2012.

In the following, for each experiment, we provide:

- one plot (a) representing the median solving time of the different PMaxSAT solvers over the benchmark under consideration;
- one plot (b) representing the minimum percentage of skipped problems $PSKIPPED_{min}$, the maximum percentage of skipped problems $PSKIPPED_{max}$ and the difference between the median solving time of the slowest and the quickest solver, namely $\Delta_{TMedian} = \max(TMedian) - \min(TMedian)$.

The main findings of the experiments are that:

1. Focusing on a singular dimension, the solver that best performs on small values of the parameter may not be the best also for big values of the same parameter;
2. The solver that best performs over a certain class of UAQ problems could not be the best also for the other classes;
3. In general, the solvers exhibit the same behavior, however it may happen that a solver clearly outperforms the others.

We can conclude that, unsurprisingly, there is not a general best solver.

Plb_bigR

Figure 8.27 presents the results of the experiment over *Plb_bigR*. The median solving time seems to grow exponentially with $|P_{lb}|$ for each solver considered; however, the slopes are really different. For example, when $|P_{lb}| = 20$, the median solving time for Maxino is around 1 second, while QMaxSAT, QMaxSATuc, and WPM1-2012 reach the timeout. In addition, note that when $|P_{lb}|$ ranges from 35 to 50, the median solving time of MaxHS grows from around 15 seconds to around 225 seconds, while all the other solvers reach the timeout. The difference in performance is even more evident in Figure 8.27b: $PSKIPPED_{min}$ is 0% even when $PSKIPPED_{max}$ is 100%. Besides, $\Delta_{TMedian}$ is almost 10 minutes when $20 \geq |P_{lb}|$.

Another interesting fact is that the solver that best performs for small values of $|P_{lb}|$ and the solver that best performs for high values of $|P_{lb}|$ are not the same: in fact, when $|P_{lb}| = 5$ the median solving time is around 20 milliseconds for Maxino and around 250 milliseconds for MaxHS; however, when $|P_{lb}| = 50$ the median solving time is around 200 seconds for MaxHS, while Maxino reached the timeout.

R_bigPlb

Figure 8.28 presents the results of the experiment over *R_bigPlb*. Even if all the solvers reach the timeout when $|R| = 60$, before this value performance differs from solver to solver. For example, when $|R| = 10$, the median solving time is 37 milliseconds for QMaxSATuc and almost 47 seconds for LMHS, while WPM1-2012 even reaches the timeout.

The difference in performance is even more evident in Figure 8.28b. $\Delta_{TMedian}$ is 10 minutes (or close) until $|R| = 30$: for this value, $PSMKIPPED_{max}$ is 1, in fact WPM1-2012 reaches the timeout, while the median solving time for Open-WBO-RES is around 15 seconds. $\Delta_{TMedian}$ then decreases towards 0, when all of the solvers reaches the timeout. Besides, $PSKIPPED_{min}$ is 0% until $|R| = 40$, while $PSKIPPED_{max}$ is 100% for all the values of $|R|$.

RPhat_bigPlb

Figure 8.29 presents the results of the experiment over *RPhat_bigPlb*. All the solver exhibit exponential growth. However, contrary to the cases analyzed above, it seems that there is a general best solver: in fact, Maxino performs best almost for all the values of \widehat{R}_P for which we have evidence, even if Open-WBO-RES could show comparable performance for higher values of the parameter. As shown in Figure 8.29b, $\Delta_{TMedian}$ rapidly increases from around 36 seconds when $\widehat{R}_P = 6$ to 10 minutes when $\widehat{R}_P = 7$. In fact, for this value, WPM1-2012 reaches the timeout for 70% of the instances.

Pub

Figure 8.30 presents the results of the experiment over *Pub*. The shape of the curves representing the median solving time is similar for all the solvers. When $|P_{ub}| = 700$, all the solvers reached the timeout. However, $\Delta_{TMedian}$ is almost 10 minutes when $|P_{ub}| = 600$. In fact, for this value, QMaxSAT skips all the instances, while Loandra can solve all the instances before the timeout.

As for *Plb_bigR*, it sometimes happens that a solver that performs well for small values of $|P_{ub}|$ is not among the best solvers for bigger values of the parameter. For example, Maxino is one of the best performing solvers when $|P_{ub}| \leq 400$, but it is also one of the worst when $|P_{ub}| \geq 500$. It is also interesting to note that, for these values of the parameter, the best performing solver is WPM1-2012, which actually is the oldest solver among the ones considered in these experiments.

C_bigR

Figure 8.31 presents the results of the experiment over *C_bigR*. Also in this case, the shape of the curves representing the median solving time of the different solvers is rather similar. In fact, for all the solvers, the median solving time is less than 100 milliseconds when $|C| \leq 40$, then it quickly grows to reach the timeout, for some solvers when $|C| = 70$ and for others when $|C| = 80$. This is also reflected in $\Delta_{TMedian}$, whose value is less than 200 milliseconds when $|C| \leq 50$, then grows to around 193 seconds when some of the solvers reach the timeout and finally decreases to 0 when all the solvers reach the timeout.

Note that, also in this case the best solver depends on the actual value of $|C|$: in fact, when $|C| \leq 50$ Maxino and Loandra are among the most performing solvers; on the contrary, they are not among the most efficient when $|C| = 60$.

rshat_bigCt

Figure 8.32 presents the results of the experiment over *rshat_bigCt*. Also over this benchmark, the curves representing the median solving time are similar: their values are less than 100 milliseconds when $\hat{r}s \leq 15$ and quickly increase to the timeout when $\hat{r}s = 25$. QMaxSATuc is the best performing solver over the values of $\hat{r}s$ for which we have evidence, while LMHS is the least performing one when $\hat{r}s = 20$. For this value, $\Delta_{TMedian}$ is around 368 seconds, while before it is less than 50 milliseconds. Finally, $\Delta_{TMedian}$ is 0 when $\hat{r}s \geq 25$, value for which all the solvers reach the timeout for the majority of the instances.

that_bigR

Figure 8.29 presents the results of the experiment over *that_bigR*. All the solvers under examination but Maxino can solve the instances before the timeout only for the smallest value of the parameter, namely $\hat{t} = 2$. On the contrary, the median solving time of Maxino over the benchmark grows until around 12 seconds when $\hat{t} = 8$. After this value, also Maxino reaches the timeout for all the instances, like the other solvers. Consequently to this behavior, $\Delta_{TMedian}$ immediately increases from less than 200 milliseconds to around 10 minutes; then it decreases to 0 seconds when also Maxino reaches the timeout. It is then clear that Maxino is by far the best performing solver among the ones under examination over *that_bigR*.

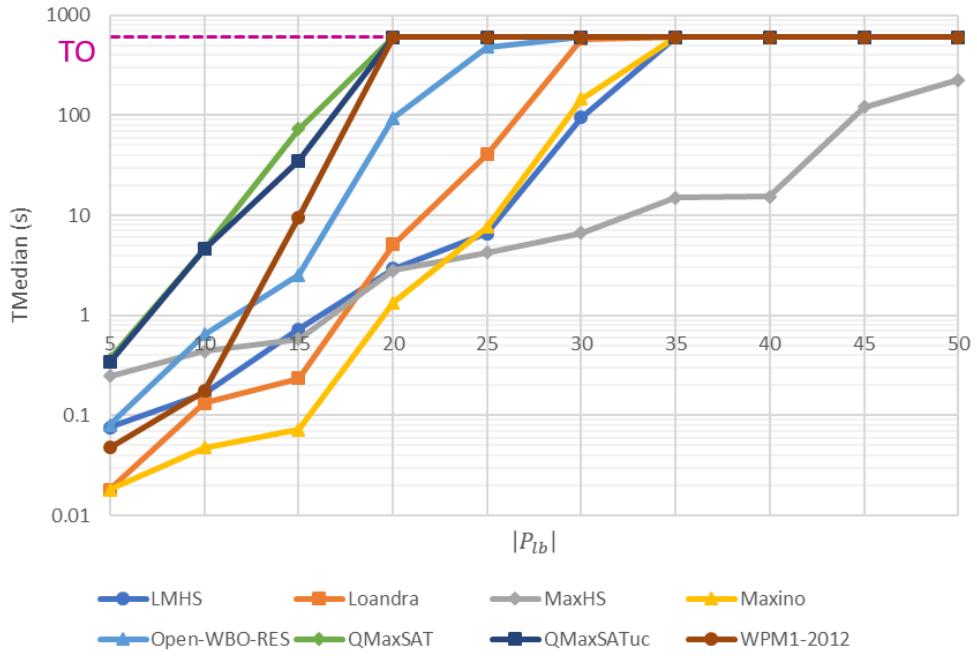
Summary

In this section, we compared different PMaxSAT solvers by running them over some of our benchmarks. From the analyses provided, it appears that:

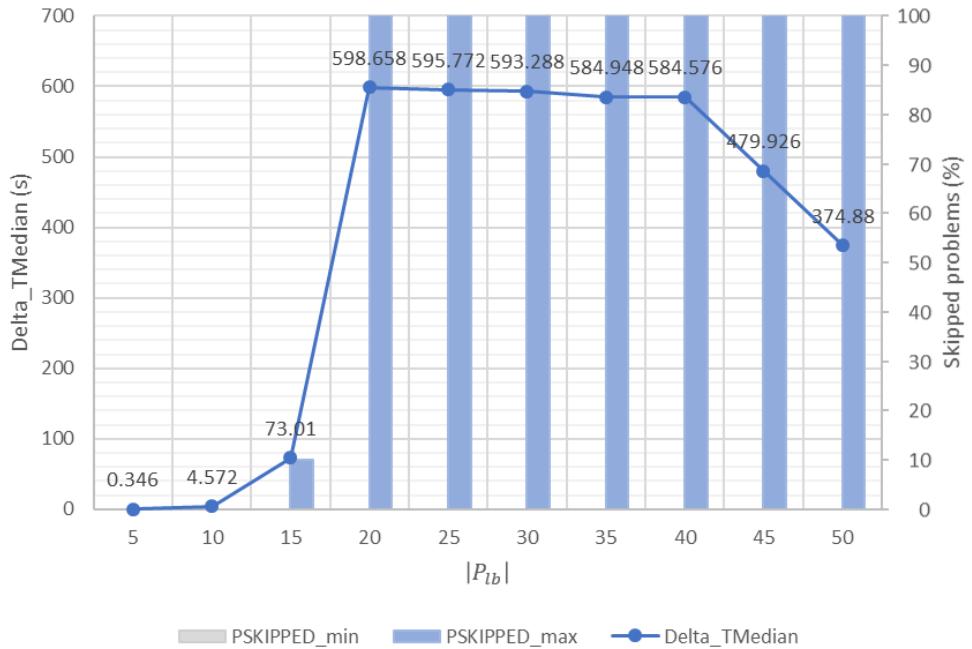
1. Focusing on a singular dimension, the solver that best performs on small values of the parameter may not be the best also for big values of the same parameter, see Maxino over *Plb_bigR*;
2. It is not necessarily true that the solver that best performs over a certain class of UAQ problems is the best also for the other classes. For example, in Figure 8.27a, for high values of $|P_{lb}|$, MaxHS is the best solver among the ones under test, while, in Figure 8.29a, the best solver is Maxino;
3. In general, the solvers exhibit the same behavior, however it may happen that a solver clearly outperforms the others, see Maxino over *that_bigR*.

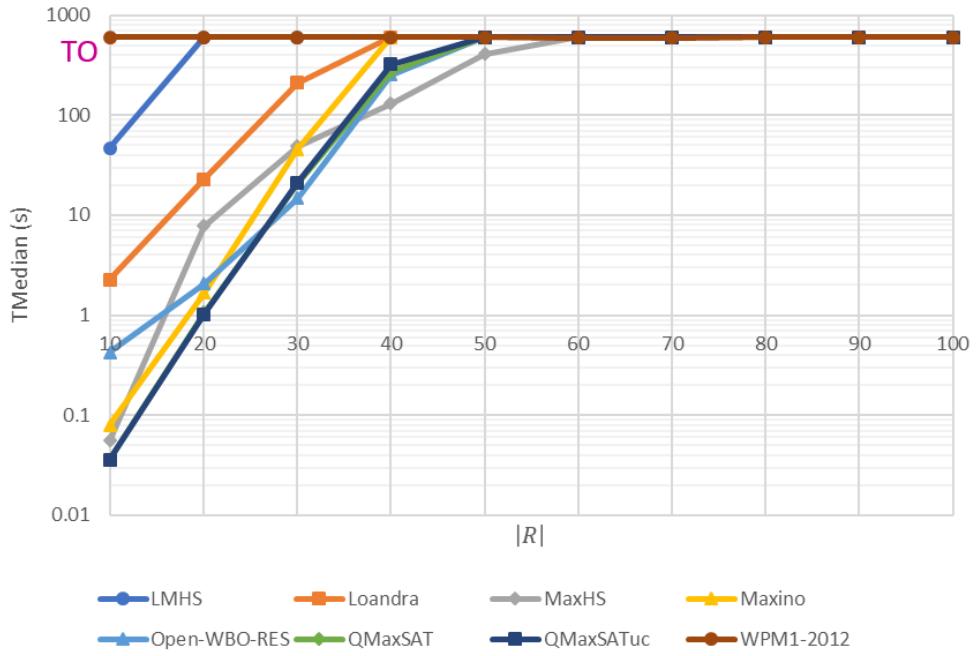
It is then clear that, for the implementation of an RBAC system, the choice of the proper solver to use must be taken carefully, according to the dimensions of the RBAC system itself. Our sets of benchmarks can be used to support this choice. In addition, UAQ-Solve makes it easy to include a new solver into the experiments: in fact, it is enough to specify the command to execute the solver.

Moreover, a possible interesting evolution of UAQ-Solve could be the integration of Machine Learning techniques to automatically select the proper solver to use according to the relevant dimensions of the UAQ problem.

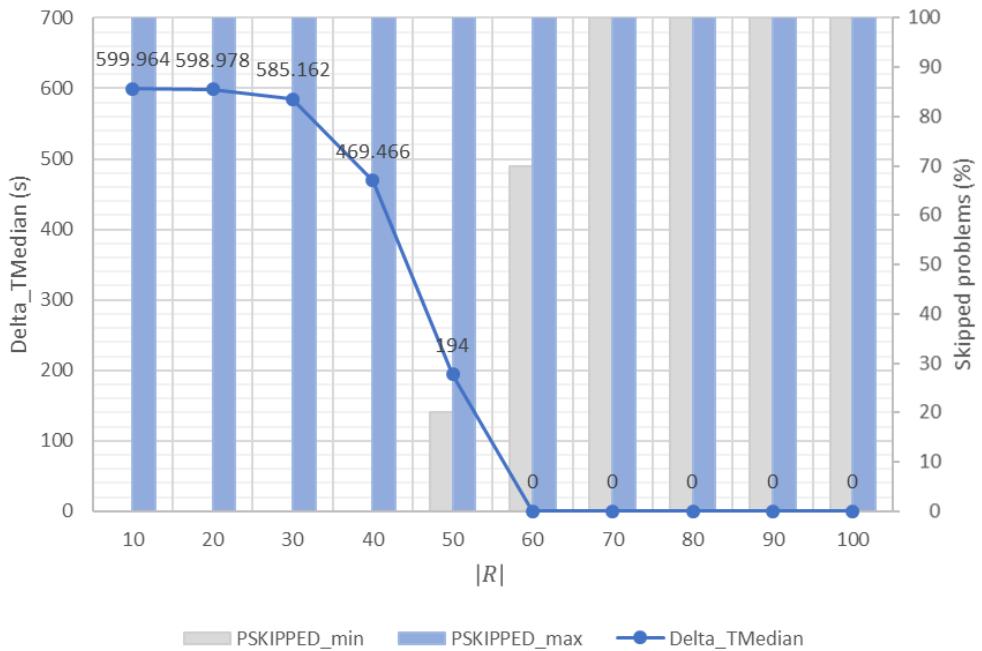


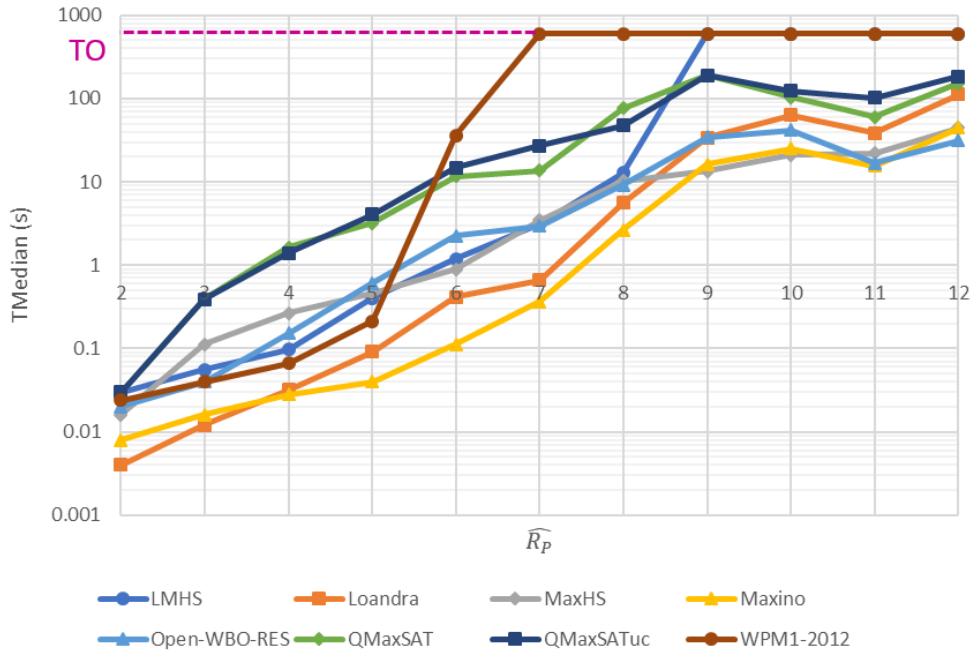
(a) Median solving time of different PMaxSAT solvers

(b) Minimum and maximum percentage of skipped problems and $\Delta_{TMedian}$ benchmarkFigure 8.27 Performance of different PMaxSAT solvers over Plb_bigR benchmark

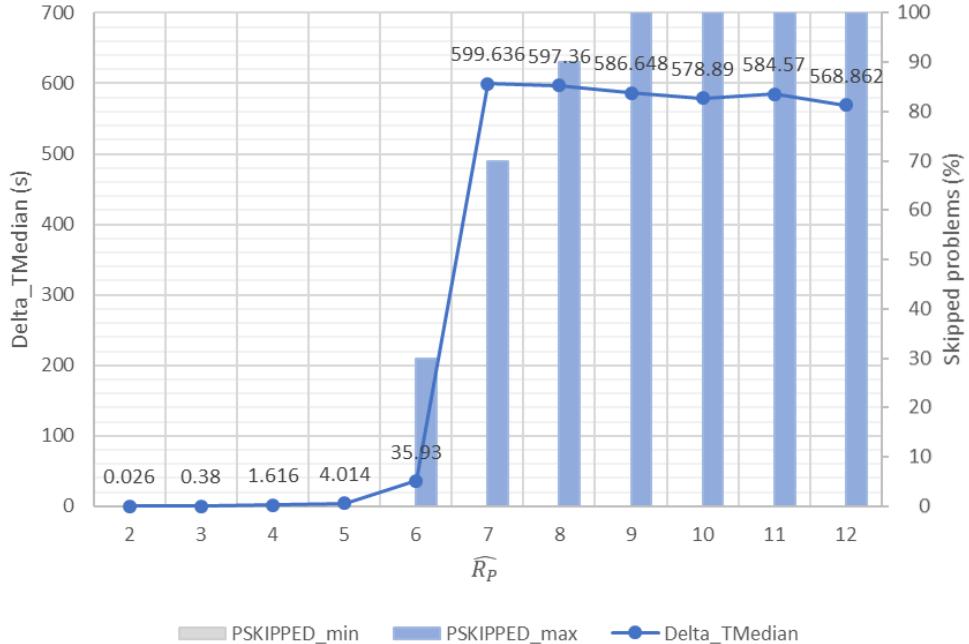


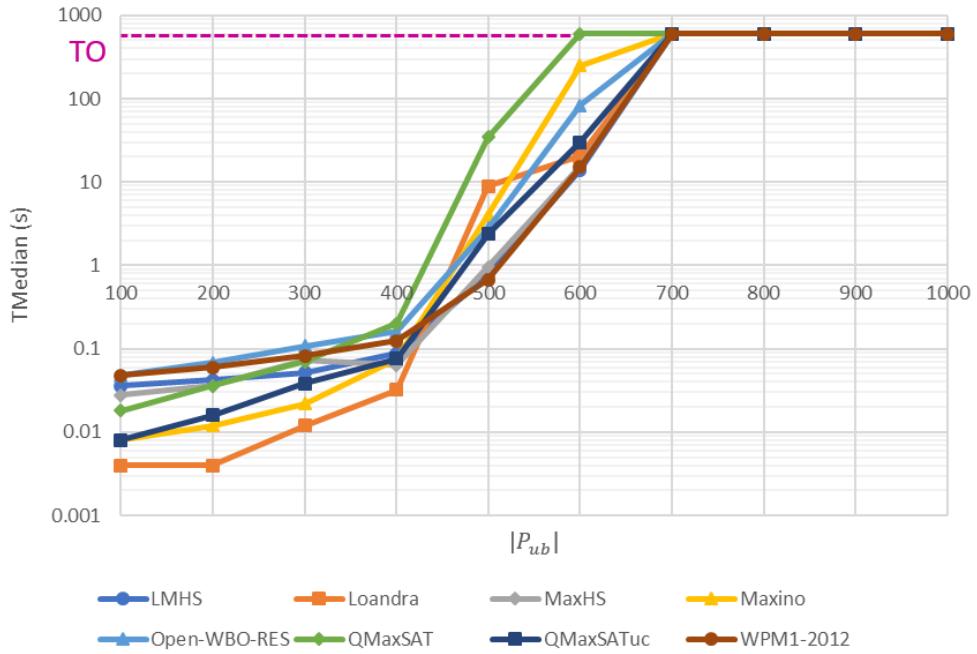
(a) Median solving time of different PMaxSAT solvers

(b) Minimum and maximum percentage of skipped problems and $\Delta_{TMedian}$ Figure 8.28 Performance of different PMaxSAT solvers over R_bigPlb benchmark

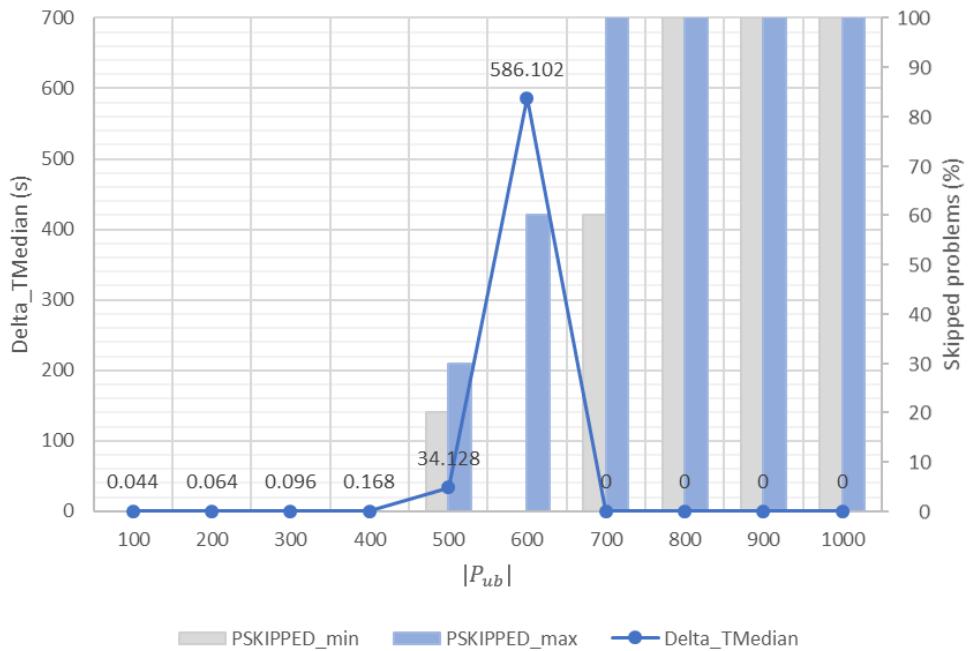


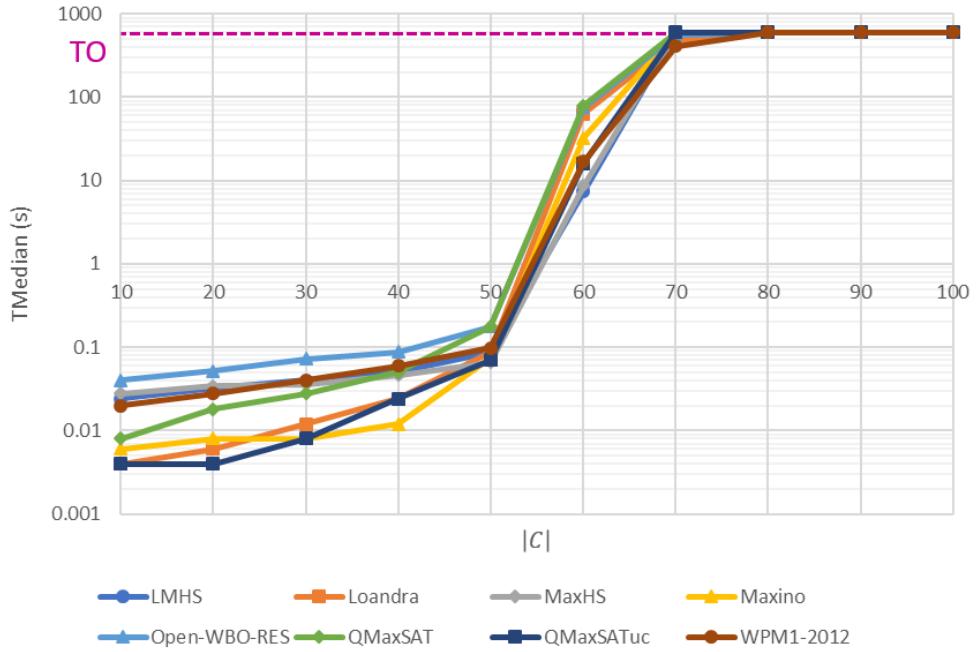
(a) Median solving time of different PMaxSAT solvers

(b) Minimum and maximum percentage of skipped problems and $\Delta_{TMedian}$ Figure 8.29 Performance of different PMaxSAT solvers over $RPhat_bigPlb$ benchmark

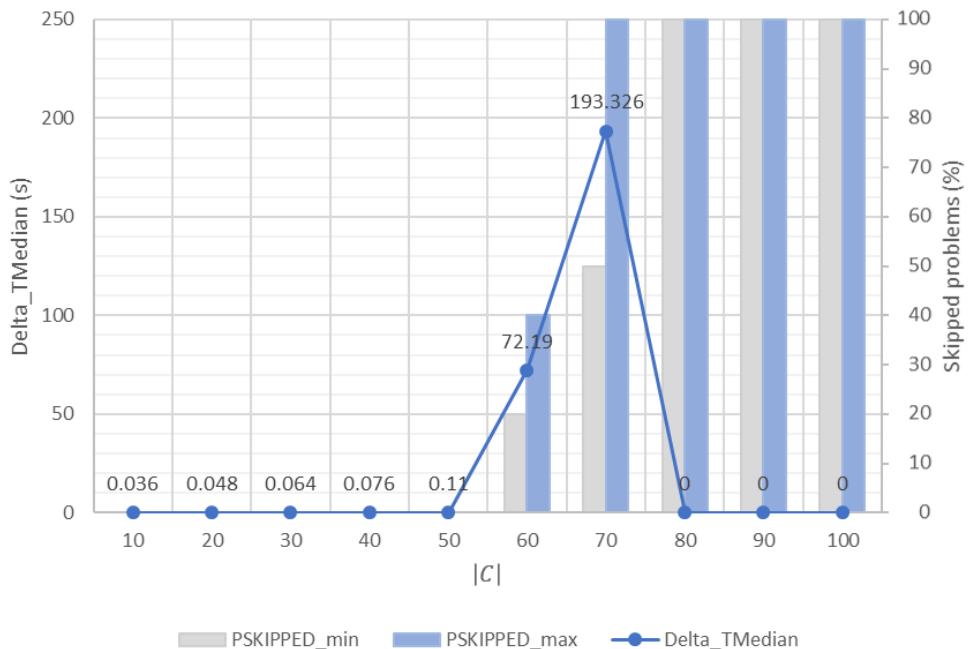


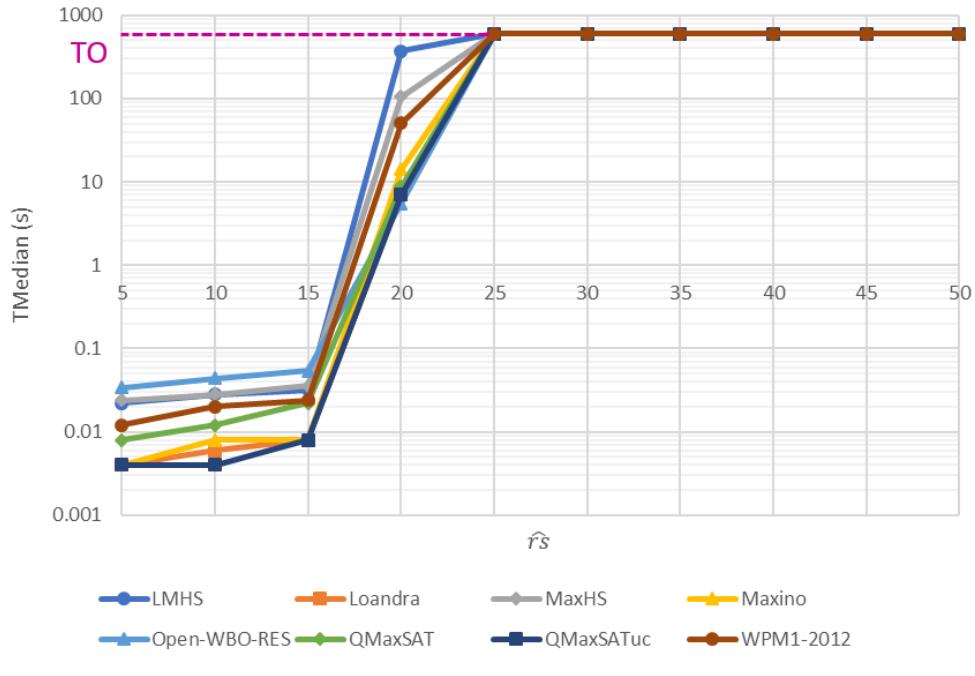
(a) Median solving time of different PMaxSAT solvers

(b) Minimum and maximum percentage of skipped problems and $\Delta_{TMedian}$ Figure 8.30 Performance of different PMaxSAT solvers over *Pub* benchmark

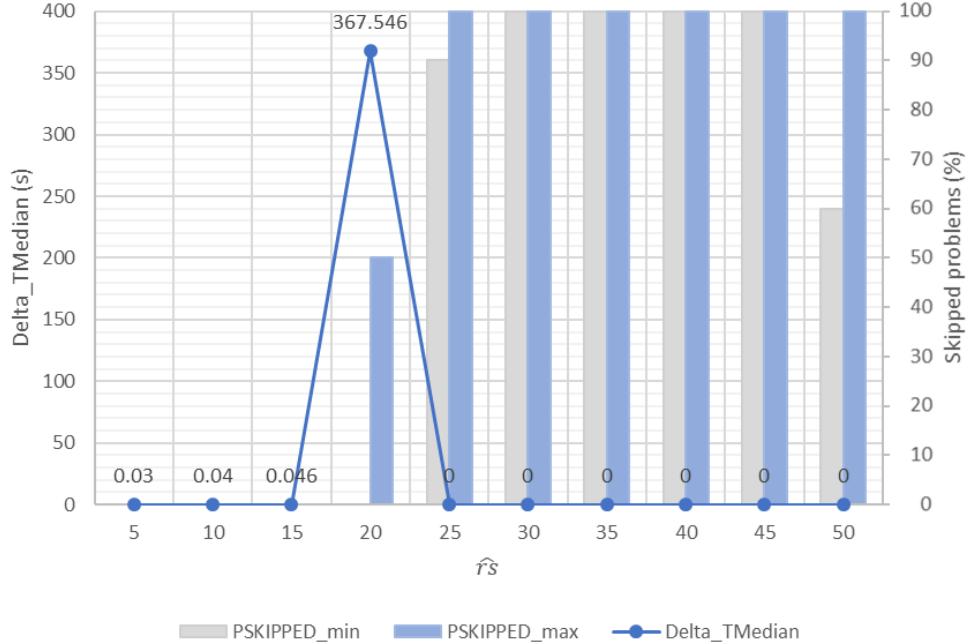


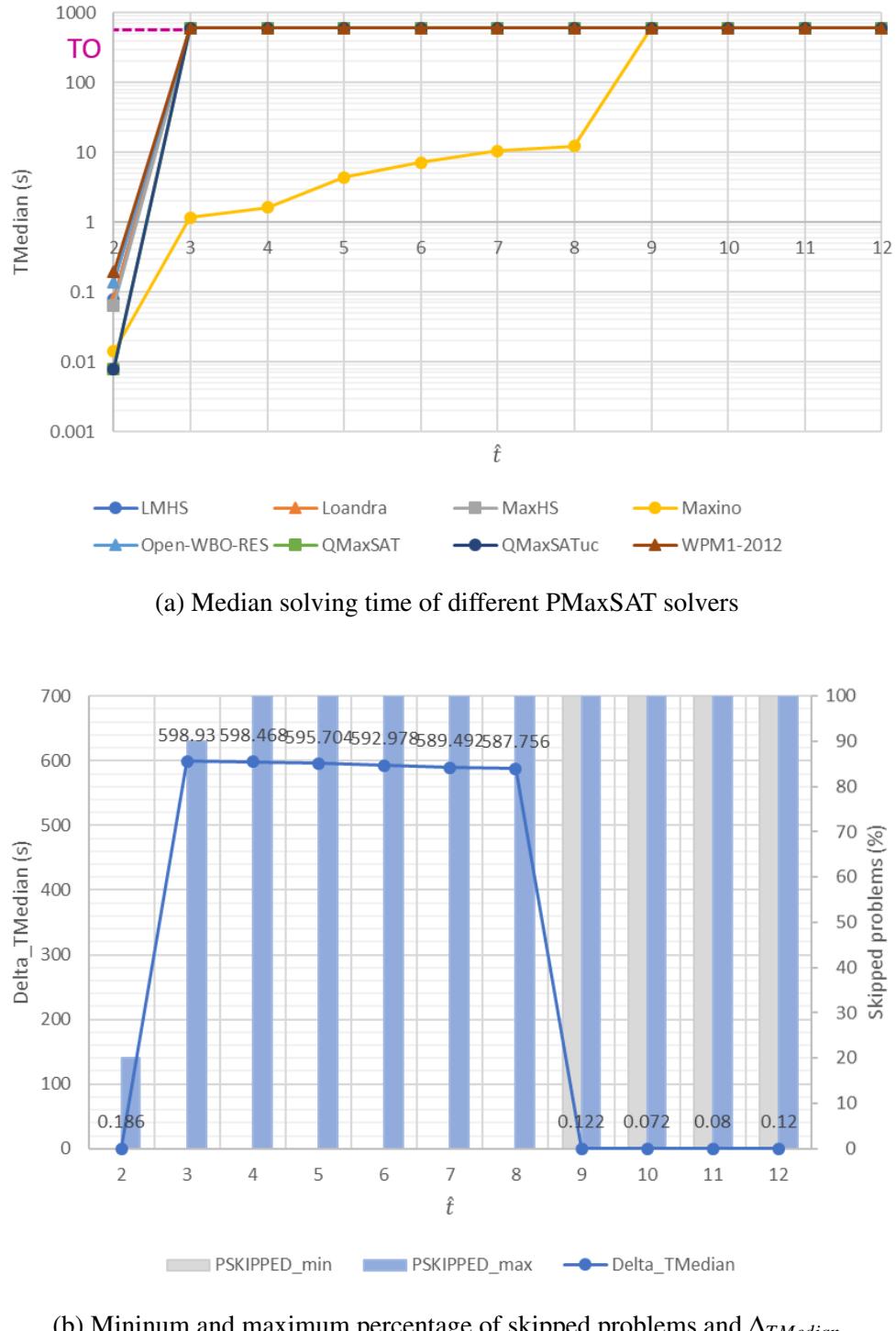
(a) Median solving time of different PMaxSAT solvers

(b) Minimum and maximum percentage of skipped problems and $\Delta_{TMedian}$ Figure 8.31 Performance of different PMaxSAT solvers over C_bigR benchmark



(a) Median solving time of different PMaxSAT solvers

(b) Minimum and maximum percentage of skipped problems and $\Delta_{TMedian}$ Figure 8.32 Performance of different PMaxSAT solvers over *rshat_bigCt* benchmark

Figure 8.33 Performance of different PMaxSAT solvers over *that_bigRPub* benchmark

8.5 Use of incomplete solvers

In Section 6.3 we presented two sets of benchmarks, one for $o_p = \min$ and the other for $o_p = \max$. Both of them contain easy benchmarks as well as hard benchmarks. In Sections 8.2 and 8.3 we used our methodology to evaluate them. In particular, the experimental results demonstrated that they are empirically adequate.

On the one hand, the results provided are good and constitute an important contribution, since to our knowledge no one before provided benchmarks evaluated systematically; on the other hand, having regard to the long time needed to solve some hard instances, one could ask if this approach can be really used in practice: a user can not wait for 10 minutes to have the authorization to access a certain resource she is legally requesting. For this reason, it is necessary to reach a compromise between the security objective (least privilege or availability) and usability. The so-called incomplete solvers already introduced in Section 7.1 can come to the aid: we remember that if at the preset timeout the incomplete solver did not find the optimum solution, it simply returns the best solution found. In this section, we focus on the benchmarks designed for $o_p = \min$ encoded with permission-based optimization.

Section 8.5.1 shows the results of the execution of the incomplete solver LMHS-inc over hard benchmarks with small timeouts: in particular, we analyze the quality of the solutions found using the optimum solutions found by MaxHS (complete solver) as a comparison. Here, the main findings are that:

- The cost of the best solution found comes closer to the cost of the optimum solution with the increase of the timeout (see Figure 8.34a);
- The incomplete solver seems to be able to quickly find a good solution, then it takes more time to converge to the optimum one;
- Considering the time spent by any complete solver to find the optimum solution, the error of the solution found by an incomplete solver in 1 second is acceptable;
- It may happen that any incomplete solver finds a solution that minimizes the solution cost, but that it does not have enough time to prove that it actually is an optimum solution;
- Although any incomplete solver could provide a sub-optimal solution, in terms of solution cost (namely, in case of $o_p = \min$, safety), the use of any incomplete solver is still more convenient than using a complete solver with $o_p = \text{any}$.

In Section 8.5.2, we compare the performance of some incomplete solvers with the performance of the equivalent complete versions over hard benchmarks. The results of the experiment show that incomplete solvers could be less performing than the complete ones over hard benchmarks and with high timeouts (such as 600 seconds, in this case).

Finally, in Section 8.5.3, we present the performance of the incomplete solver Loandrainc in comparison with the equivalent complete version. According to the results, the performance of a complete and an incomplete solver seem to be comparable when executing over easy benchmarks.

8.5.1 Incomplete solvers over hard instances with a small timeout

In the following, we present the results of the execution of LMHS-inc over *Plb_bigR*, *R_bigPlb*, and *RPhat_bigPlb* with $o_p = \text{min}$, $o_r = \text{any}$, and $pri = p$. For each benchmark, we run LMHS-inc with different timeouts, namely 1, 2, 3, 4 and 5 seconds. In order to evaluate LMHS-inc performance, we define the following quantities parametric in the parameter under consideration in the benchmark (x)⁸:

$$\Delta_{\text{cost}}(x) = \text{Median}_i(\text{Cost}_{\text{inc}}(x_i) - \text{Cost}_{\text{opt}}(x_i)),$$

where $\text{Cost}_{\text{inc}}(x_i)$ and $\text{Cost}_{\text{opt}}(x_i)$ are respectively the cost of the solution found by LMHS-inc and by MaxHS for the i^{th} instance in x ,

$$\varepsilon(x) = \text{Median}_i\left(\frac{\text{Cost}_{\text{inc}}(x_i) - \text{Cost}_{\text{opt}}(x_i)}{\text{Cost}_{\text{opt}}(x_i)}\right),$$

$$\varepsilon/\Delta_T = \text{Median}_i\left(\frac{\text{Cost}_{\text{inc}}(x_i) - \text{Cost}_{\text{opt}}(x_i)}{\text{Cost}_{\text{opt}}(x_i) \times (T_{\text{opt}}(x_i) - T_{\text{inc}}(x_i))}\right),$$

where $T_{\text{opt}}(x_i)$ and $T_{\text{inc}}(x_i)$ are the solving time of MaxHS and LMHS-inc respectively over the i^{th} instance in x ($T_{\text{inc}}(x_i)$ is at most equal to the timeout),

$$\text{Cost}_{\text{opt}}(x) = \text{Median}_i(\text{Cost}_{\text{opt}}(x_i)),$$

$$\text{Cost}_{\text{inc}}(x) = \text{Median}_i(\text{Cost}_{\text{inc}}(x_i)),$$

⁸In the following, with $f(x) = \text{Median}_i(g(x_i))$ we mean that, when $x = x_0$, $f(x_0)$ is the median of the values of g calculated in the 10 instances of the benchmark characterized by $x = x_0$.

$$\Delta_{any}(x) = \text{Median}_i(\text{Cost}_{any}(x_i) - \text{Cost}_{inc}(x_i)),$$

where $\text{Cost}_{any}(x_i)$ is the cost of the solution found by MaxHS with $o_p = \text{any}$ over the i^{th} instance in x , and

$$\varepsilon_{any}(x) = \text{Median}_i\left(\frac{\text{Cost}_{any}(x_i) - \text{Cost}_{inc}(x_i)}{\text{Cost}_{any}(x_i)}\right).$$

The use of all the aforementioned quantities enables to have a general overview of the performance of the incomplete solver under examination.

In particular, Δ_{cost} suggests the distance among the solution of the complete solver and the optimum solution, in terms of cost. We have to remember that, when our objective is the least privilege, the more the cost is high, the more we are exposed to high risk.

Instead, ε represents the error of the solution provided by LMHS-inc. ε/Δ_T is another important value, because, differently from ε , it also considers the time spent by MaxHS to find the optimum solution. In fact, since the UAQ problem complexity increases with the parameter under consideration, we expect that ε increase with the parameter: namely, the increase in ε could not be due to the bad performance of LMHS-inc, but to the increase of the problem complexity.

As regards Cost_{inc} and Cost_{opt} , we expect that Cost_{inc} decreases and get closer to Cost_{opt} with the increase of the timeout.

Finally, Δ_{any} and ε_{any} are used to show the benefit we have by using LMHS-inc with respect to MaxHS with no optimization: in fact, even when LMHS-inc is not able to find an optimum solution, the best solution found by the incomplete solver still could be better, in term of cost, than any solution found without any particular objective.

For each benchmark, we show 5 figures:

1. One plot showing Δ_{cost} for each timeout used, namely 1, 2, 3, 4 and 5 seconds;
2. One plot showing ε for each timeout used;
3. One plot showing ε and ε/Δ_T with timeout 1 second and the median solving time of the complete solver;
4. One plot showing Cost_{inc} with the increase of the timeout and Cost_{opt} ;
5. One plot showing Δ_{any} and ε_{any} with timeout 1 second. For all the experiments, MaxHS with $o_p = o_r = \text{any}$ could solve all the problems in less than 40 milliseconds.

In the following subsections, we show the results of the experiments over the different benchmarks.

Plb_bigR

The results of the experiment over *Plb_bigR* are presented in Figure 8.34. In particular, Figure 8.34a presents Δ_{cost} for LMHS-inc with different timeouts. All the plots have the same shape and increase with the increasing of $|P_{lb}|$. This behavior is not surprising: in fact, for the smaller values of $|P_{lb}|$, namely when $|P_{lb}| \leq 15$, LMHS-inc is able to find an optimum solution regardless of the timeout t . Then, the incomplete solver is not able to find the optimum solution in 1 second, in fact the related plot starts increasing. However, LMHS-inc still can find the optimum solution until $|P_{lb}| = 30$ with the other timeouts. After that value, Δ_{cost} increases in all the cases, even if with different slopes. It is interesting to note that LMHS-inc finds a reasonably good solution for all the problems already after 1 second, then Δ_{cost} visibly improves when the timeout is 2 seconds: for example, when $|P_{lb}| = 50$, Δ_{cost} is 14 after 1 second and 7.5 after 2 seconds. After 2 seconds, the improvement of the solutions is slower. This is reflected in ε , in Figure 8.34b, where the improvement of solution from $t = 1$ to $t = 2$ is tangible: in this range, ε goes from around 0.12 to around 0.65.

This behavior is particularly clear also in Figure 8.34d, which shows the trend of the median cost of the solutions found by LMHS-inc with the increasing of the timeout for $|P_{lb}| = 50$: $Cost_{inc}$ rapidly decreases from 128 for $t = 1$ to 120.5 for $t = 3$, then it decreases more smoothly towards $Cost_{opt}$, which is 116.

Focusing on the best solution found after 1 second, Figure 8.34c shows that ε reaches around 0.12 when $|P_{lb}| = 50$, which is not a bad result, considering that MaxHS takes more than 200 seconds to find an optimum solution. In fact, ε/Δ_T is less than 0.001. Here we also observe that ε/Δ_T decreases with the increase of $|P_{lb}|$.

With regard to the comparison with the solutions found by MaxHS with $o_p = any$, as visible in Figure 8.34e, Δ_{any} decreases from 364 to 222, while ε_{any} decreases from around 0.92 to around 0.63. This is logical, because the problem becomes more complex with the increase of $|P_{lb}|$. However, when $|P_{lb}| = 50$, Δ_{any} is still 222, which is a relevant value.

Plb_bigR contains 100 instances. LMHS-inc reaches the timeout of 1 second over 71 of them. It is very interesting that for 12 of them (having $15 \leq |P_{lb}| \leq 35$) the cost is the same as the optimum solutions found by MaxHS. This means that the solver finds an optimum solution before the timeout, but it does not have enough time to prove it.

R_bigPlb

Figure 8.35 shows the results of the experiment over *R_bigPlb*. Also in this case, the Δ_{cost} plots (in Figure 8.35a), as well as the ε plots (in Figure 8.35b), have the same shape. However, differently from the previous experiment, there is not an important improvement of the solution from $t = 1$ second to $t = 2$ seconds. In particular, it seems that LMHS-inc easily finds a reasonably good solution for all the instances and for any value of $|R|$ just after one second (with ε less than 0.1 when $|R| = 50$), then it is harder for it to find better solutions. This is confirmed by $Cost_{inc}$ in Figure 8.35d, which is the median cost of the solutions found by the incomplete solver over the instances characterized by $|R| = 40$. The convergence of $Cost_{inc}$ to $Cost_{opt}$ is slow: in fact, $Cost_{inc}$ decreases from 258.5 to 252 with $|R|$ ranging from 10 to 50. However, ε (in Figure 8.35c) is still less than 0.1 when $|R| = 40$, while ε/Δ_T is less than 0.001. In particular, ε/Δ_T decreases from around 0.001 when $|R| = 20$ to around 0.0002 when $|R| = 40$. This is due to the fact that the median solving time of MaxHS increases from 56 milliseconds to 406 seconds with $|R|$ ranging from 10 to 50.

Finally, Figure 8.35e shows that Δ_{any} and ε_{any} increase, the former from 12 to 51 and the latter from 0.04 to 0.17, confirming that it is still more convenient to use an incomplete solver than a complete solver with no optimization objective.

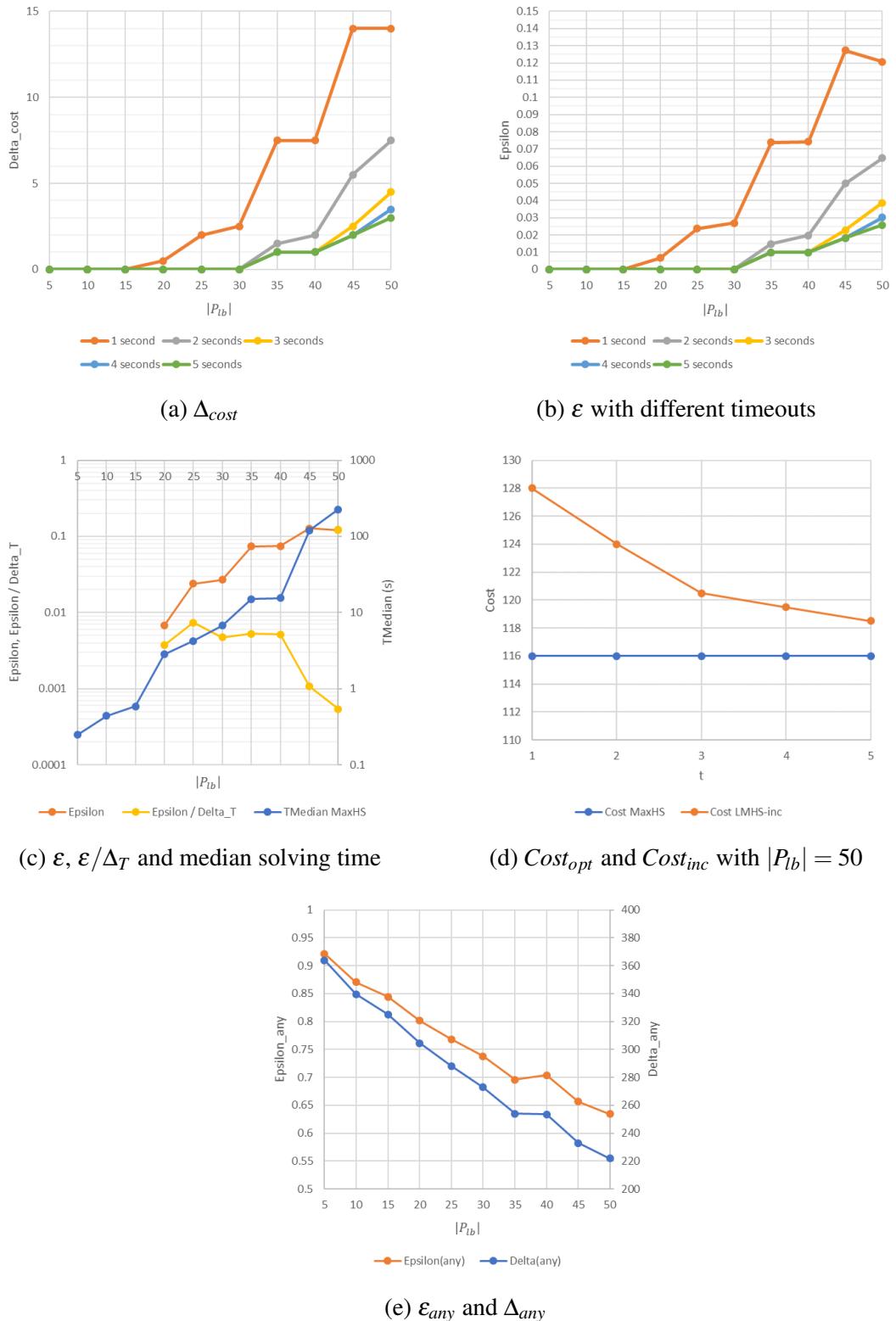
RPhat_bigPlb

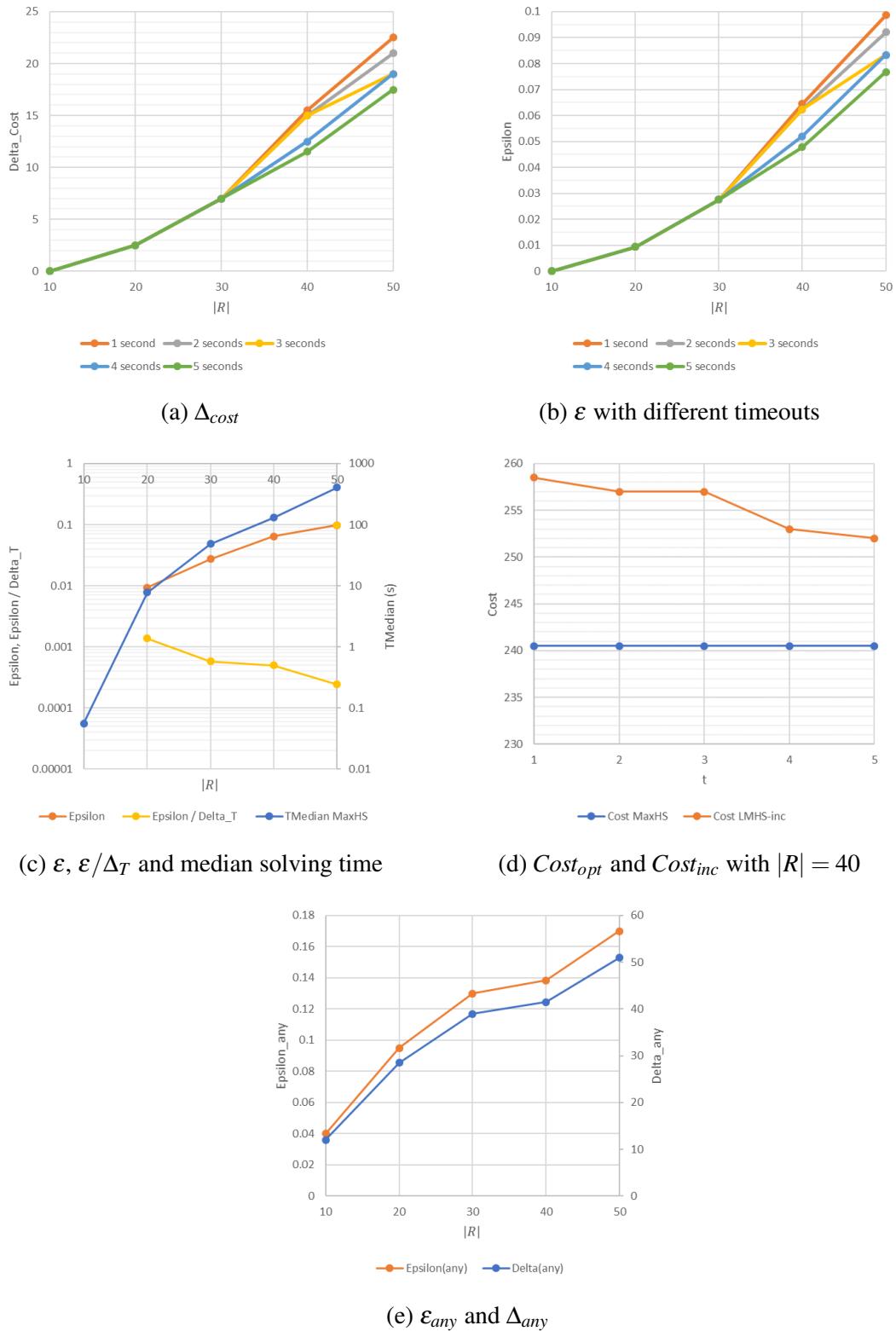
Figure 8.36 presents the results of the experiments over *RPhat_bigPlb*. As shown in Figure 8.36a, when $\widehat{R}_P \leq 7$ LMHS-inc is able to find an optimum solution before the timeout, in fact Δ_{cost} for $t = 1$ second is 0. Then, Δ_{cost} increases to 3 when $\widehat{R}_P = 9$ and to 7 when $\widehat{R}_P = 12$. ε (Figure 8.36b) behaves similarly, reaching a maximum value of around 0.09 when $\widehat{R}_P = 12$. The best solution found by the incomplete solver improves with $t = 2$ seconds, for which Δ_{cost} and ε have a similar shape as in the case of $t = 1$ second. However, the maximum values for Δ_{cost} and ε are respectively 3 and 0.04. The solution further improves with $t = 3$. Finally, LMHS-inc can find an optimum solution for all the instances proposed when $t = 4$ and $t = 5$. The improvement of the solution with the increase of the timeout is also visible in Figure 8.36d, which shows $Cost_{inc}$ for $\widehat{R}_P = 12$. In particular, $Cost_{inc}$ decreases from 85.5 when $\widehat{R}_P = 2$ to 78 when $\widehat{R}_P = 12$, thus arriving very close to $Cost_{opt}$, which is 77.5.

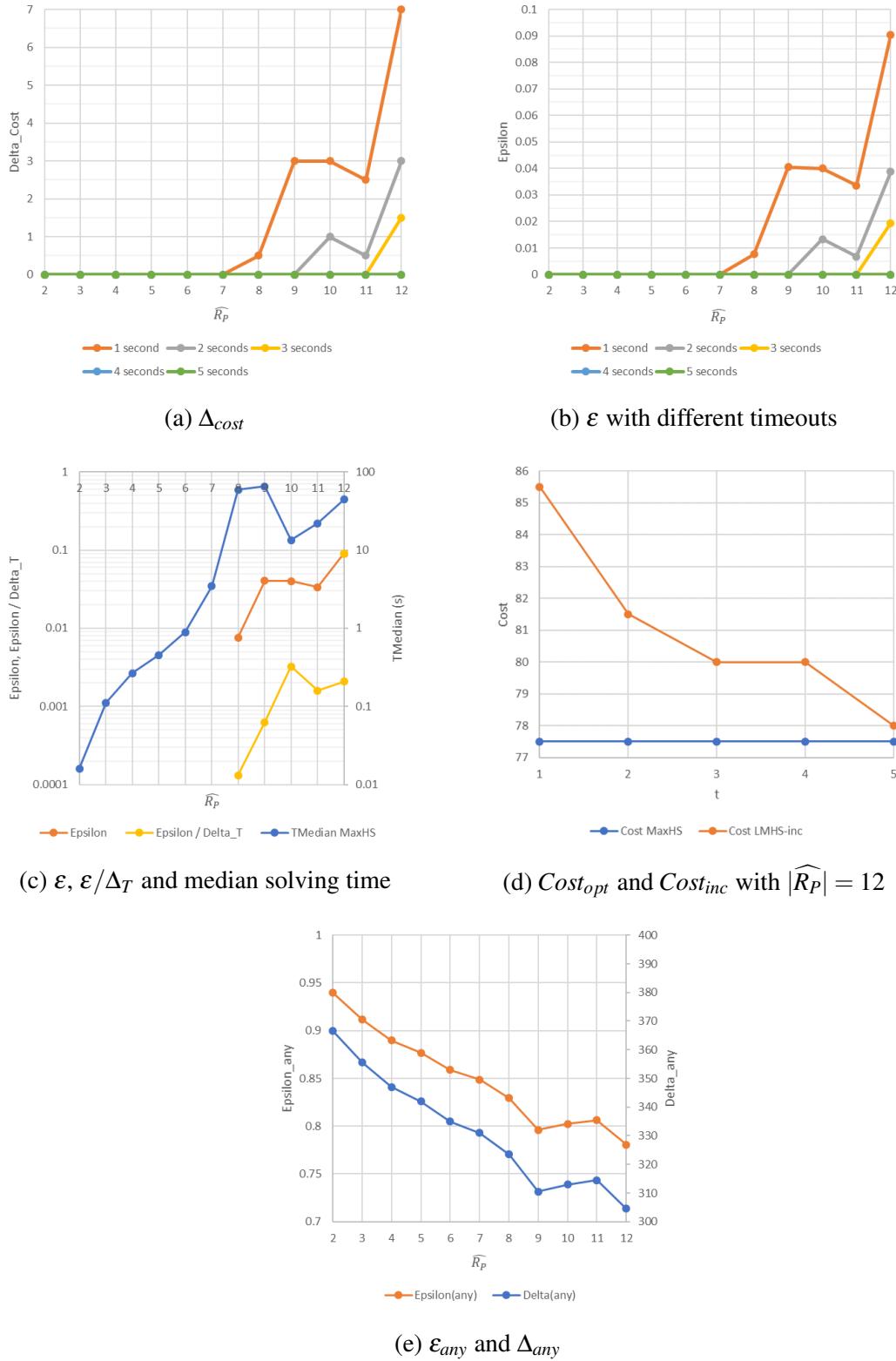
Figure 8.36c shows that the quality of the solutions found by the incomplete solver is acceptable also after 1 second: in fact, ε is at most around 0.1 when $\widehat{R}_P = 50$. This is a good result, considering that MaxHS needs around 44 seconds to find the optimum solution. Consequently, also ε/Δ_T is low, around 0.002.

Finally, Figure 8.36e shows the benefit of using LMHS-inc with $t = 1$ in comparison to MaxHS with $o_p = \text{any}$. Both Δ_{any} and ε_{any} decrease, the former from 366.5 to 304 and the latter from around 0.94 to around 0.78 with \widehat{R}_P ranging from 2 to 12. Even if the values of Δ_{any} and ε_{any} decrease with the increase of the parameter under consideration, their values are still important when $\widehat{R}_P = 12$.

Rphat_bigPlb contains 110 instances. LMHS-inc reaches the timeout of 1 second over 53 of them. It is very interesting that 16 of them (having $\widehat{R}_P \geq 7$) have the same cost provided by the complete solver, meaning that they actually are optimum. Even more interesting is that MaxHS over them takes from around 4 seconds to around 30 seconds to find an optimum solution. This means that sometimes it is far easier to find an optimum solution than proving that it actually is optimum.

Figure 8.34 Comparison between MaxHS and LMHS-inc over Plb_bigR

Figure 8.35 Comparison between MaxHS and LMHS-inc over R_bigPlb

Figure 8.36 Comparison between MaxHS and LMHS-inc over $RPhat_bigPlb$

8.5.2 Incomplete solvers over hard instances with timeout 600 seconds

In Section 8.4 we used our benchmarks to evaluate some complete solvers. For the sake of completeness, here we repeat the same experiments by running some incomplete solvers, in order to compare their performance with the performance of the respective complete version. In particular, here we evaluate the following incomplete solvers: LMHS-inc, Loandra-inc, MaxHS-inc, Open-WBO-LSU, QMaxSAT-inc, and QMaxSATuc-inc. To this aim, we execute them over the benchmarks Plb_bigR , R_bigPlb , and $RPhat_bigPlb$ with the same timeout used during the evaluation of the complete solvers, namely 600 seconds. The results of this comparison are shown in Figure 8.37. For each benchmark, we show:

- T_{Median} , namely the median solving time of the incomplete solvers under examination;
- $\Delta_{T_{Median}}$, namely the difference among the median solving time of the incomplete solver and the median solving time of the respective complete version.

Every incomplete solver over the three benchmarks exhibits similar behavior to the respective complete solver. When running over Plb_bigR (Figures 8.37a and 8.37b), LMHS-inc, QMaxSAT-inc and QMaxSATuc-inc perform almost like the complete versions: in particular, LMHS-inc only takes around 12 seconds more than LMHS when $|P_{lb}| = 30$; in particular, T_{Median} is around 107 seconds, compared to the 94 seconds for LMHS. When $|P_{lb}| = 15$, QMaxSATuc-inc takes around 25 seconds more than QMaxSATuc, which solves the instances in around 35 seconds. However, for the same value of the parameter, QMaxSAT-inc performs better than QMaxSAT, spending around 13 seconds less. Instead, MaxHS-inc outperforms MaxHS for almost all the cases: in particular, it takes around 26 and 36 seconds less than MaxHS when $|P_{lb}| = 45$ and $|P_{lb}| = 50$ respectively. On the contrary, when $|P_{lb}| = 25$, $\Delta_{T_{Median}}$ presents a peak for Loandra-inc and Open-WBO-LSU, around 110 seconds the former and around 120 seconds the latter. These peaks are due to the fact that the incomplete solvers reach the timeouts before the complete ones. In fact, later T_{Median} is again 0 when also the complete solvers reach the timeout.

When running over R_bigPlb , the only solver which does not perform worse than the related complete solver is LMHS; however, performance is not good in both the cases. MaxHS-inc performs worse than MaxHS when $30 \leq |R| \leq 60$, then they both reach the timeout. Like in the previous case, $\Delta_{T_{Median}}$ reaches a maximum for Loandra-inc, Open-WBO-LSU, QMaxSAT-inc, and QMaxSATuc-inc: in particular, the maximum values are around 300 seconds, which is high considering the corresponding values of T_{Median} . Again, the peaks are due to the fact the incomplete solvers reach the timeout before complete ones.

The results of the experiment over *RPhat_bigPlb* is surprising: in fact, even if the solvers under examination do not reach the timeout, *TMedian* is not monotonic for most of the solver. In particular, QMaxSAT-inc and QMaxSATuc-inc sometimes outperform QMaxSAT and QMaxSATuc respectively, sometimes the opposite situation is verified. On the contrary, Loandra-inc performs worse and worse than Loandra when $|R| \geq 8$. Differently from the others, LMHS-inc is the only solver that behaves exactly like the respective complete version.

It is then clear that there are situations in which it is more convenient to use complete solvers and others in which incomplete solvers are more suitable. However, we have to consider that:

- The incomplete solver "terminates only when it is able to prove that its best model is in fact optimal. However, often it is able to find very good upper bounding models or even optimal models long before termination (proving a model to be optimal is generally as hard or even harder than finding it)"⁹ (in particular, they were talking about MaxHS);
- In our context, we will never use incomplete solvers over difficult problems with high timeouts: on the contrary, we are interested in their use with a small timeout to enable usability, while preserving a certain level of safety.

8.5.3 Incomplete solvers over easy instances

Our last experiment aims at comparing the performance of the incomplete solver Loandra-inc with the performance of the complete solver Loandra. In particular, we execute the two solvers over *Plb_smallR*, *R_smallPlb*, *RPhat_medPlb*, *RPhat_smallPlb*, *Pub*, *C*, *rshat*, and *that*, which have been demonstrated to be easy benchmarks.

Figure 8.38 presents the results of the experiment: for each benchmark, we provide the median solving time of both the solvers. The results demonstrate that Loandra-inc performs similarly to (and consequently could be used in place of) Loandra over all the benchmarks.

⁹https://helda.helsinki.fi//bitstream/handle/10138/237139/mse18_proceedings.pdf?sequence=1

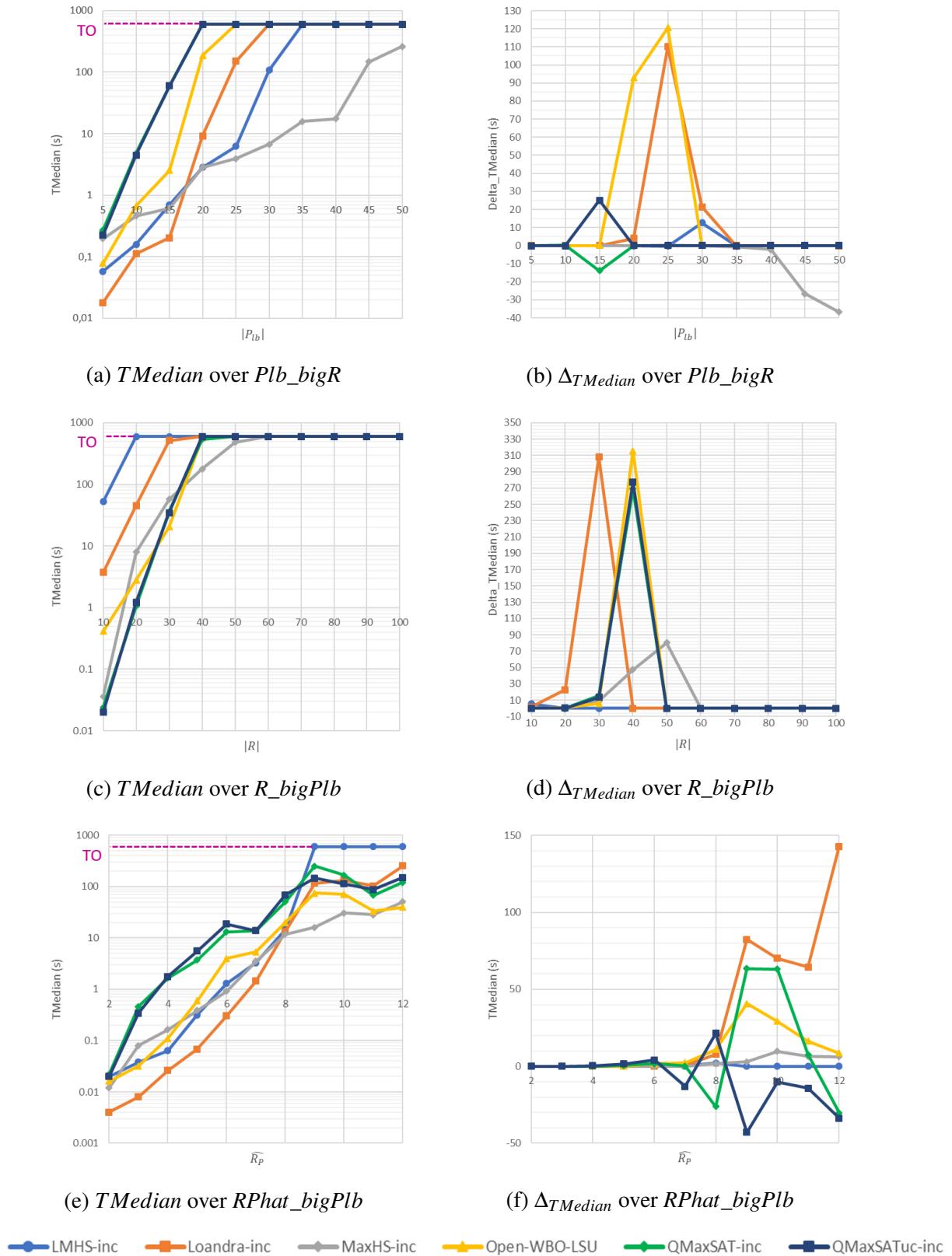


Figure 8.37 Performance of incomplete solvers over hard benchmarks with timeout 600 seconds

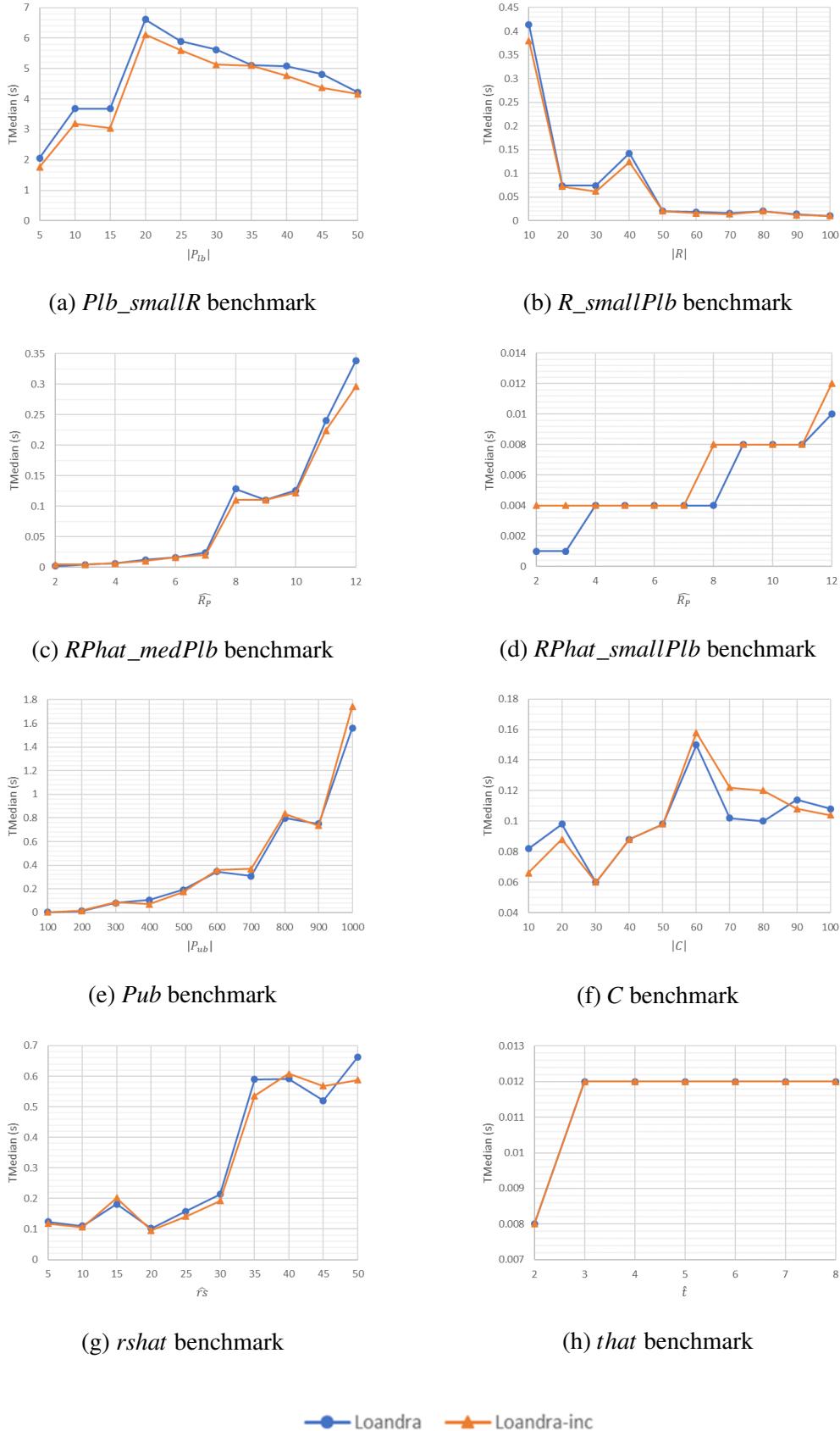


Figure 8.38 Performance of UAQ-Solve leveraging Loandra and Loandra-inc over the easy benchmarks presented in Tables 6.6 and 6.7 with permission-based optimization

8.5.4 Summary

In this section we presented the results of a number of experiments aiming at evaluating the incomplete solvers. In particular.

- We executed LMHS-inc over difficult benchmarks and provided a general overview of the quality of the best solutions provided using different timeouts (1, 2, 3, 4 and 5 seconds);
- We compared some incomplete solvers with the respective complete version with 600 seconds as timeout;
- We compared the performance of Loandra-inc and Loandra over easy benchmarks.

From the experiments it appears that:

- The more the timeout of the incomplete solver increases, the more the cost of the best solution found is close to the cost of the optimum solution;
- It may happen that any incomplete solver finds a solution that minimizes the solution cost, but that it does not have enough time to prove that it actually is the optimum solution;
- Considering the time spent by any complete solver to find the optimum solution, the solutions found by LMHS-inc in 1 second is acceptable. In particular, the error ε is always at most 0.13;
- It seems that the incomplete solver can quickly find a good solution, then it takes more time to converge to the optimum one;
- Although any incomplete solver could provide a sub-optimal solution, in terms of solution cost (namely, in case of $o_p = \min$, safety), the use of any incomplete solver is still more convenient than using a complete solver with $o_p = \text{any}$;
- Over hard benchmarks and with high timeouts (such as 600 seconds), the incomplete solvers could be less performing than the complete ones;
- The performance of Loandra-inc and Loandra is comparable when executing over easy benchmarks.

Considering the outcome of the aforementioned experiments, we can then conclude that the incomplete solvers are a valid solution to obtain a good compromise between the security objective and usability. As a consequence, we propose their use to tackle the UAQ problem for RBAC.

8.6 Summary of experimental results

This section aim at summarizing the main results presented in this chapter.

Section 8.1 demonstrated that state-of-the-art benchmarks are incomplete and inadequate to assess the effectiveness of UAQ solvers. The suite of benchmarks consists only of easy benchmarks, that cannot be used to stress-test solvers. They can only be used to check if solvers are empirically efficient over them. In fact, we used them to compare the performance of UAQ-Solve, 2D-Opt-Search and 2D-Opt-CNF. The result of the comparison is that UAQ-Solve outperformed the other solvers over all the instances. Another issue is that benchmarks for $o_p = \max$ are totally missing.

Section 8.2 contains the evaluation of our benchmarks designed through our methodology for $o_p = \min$. We evaluated the benchmarks by running UAQ-Solve over them. All the benchmarks proposed stimulated the behavior for which they were designed. We demonstrated that the suite of benchmarks designed for $o_p = \min$ is complete and empirically adequate. In fact, it consists of both hard and easy benchmarks. The former can be used to check the efficiency of the solvers. The latter can be used to stress-test the solvers. We also shown that joint optimization does not seem to be particularly detrimental for performance. We then shown that UAQ-Solve outperformed both 2D-Opt-Search and 2D-Opt-CNF over the vast majority of the benchmarks.

Section 8.3 contains the evaluation of our benchmarks designed through our methodology for $o_p = \max$. We evaluated the benchmarks by running UAQ-Solve over them. All the benchmarks proposed but one¹⁰ stimulated the behavior for which they were designed. The suite of benchmarks designed for $o_p = \max$ is complete and empirically adequate and consists of both easy and hard benchmarks. Comparing the performance over the same benchmarks encoded for permission-based and joint optimization, it turned out that the performance of UAQ-Solve deteriorates with joint optimization.

¹⁰the one parametric in $|P_{ub}|$ over which UAQ-Solve exhibited an exponential growth.

Section 8.4 compared the performance of different PMaxSAT solvers over some of our benchmarks. The experimental results shown that:

1. Focusing on a singular dimension, the solver that best performs on small values of the parameter may not be the best also for big values of the same parameter;
2. The solver that best performs over a certain class of UAQ problems could not be the best also for the other classes;
3. In general, the solvers exhibit the same behavior, however it may happen that a solver clearly outperforms the others.

Section 8.5 shown the results of the execution of incomplete solvers over our benchmarks. From the experiments it appeared that:

- The cost of the best solution found comes closer to the cost of the optimum solution with the increase of the timeout;
- The incomplete solver seems to be able to quickly find a good solution, then it takes more time to converge to the optimum one;
- Considering the time spent by any complete solver to find the optimum solution, the error of the solution found by an incomplete solver in 1 second is acceptable;
- It may happen that any incomplete solver finds a solution that minimizes the solution cost, but that it does not have enough time to prove that it actually is an optimum solution;
- Although any incomplete solver could provide a sub-optimal solution, in terms of solution cost (namely, in case of $o_p = \text{min}$, safety), the use of any incomplete solver is still more convenient than using a complete solver with $o_p = \text{any}$;
- Over hard benchmarks and with high timeouts (such as 600 seconds), the incomplete solvers could be less performing than the complete ones;
- The performance of a complete and an incomplete solver seem to be comparable when executing over easy benchmarks.

Chapter 9

Conclusions and future work

9.1 Conclusions

In this dissertation, we introduced the problem of access control to healthcare data in Health Information Systems. Healthcare data need to be adequately protected to establish trust between the actors involved in the healthcare process; at the same time, healthcare data must always be available when needed. As a consequence, usability is a fundamental feature for the system to be accepted and adopted. In this thesis, we focused on the UAQ problem for RBAC, which is key to support permission-level system-user interaction. In spite of its computational intractability, several techniques have been proposed to tackle the UAQ problem. However, none of them provides a valuable and satisfactory experimental evaluation and, as a consequence, the actual efficiency of the solutions proposed is not clear. Most of the techniques have been experimentally evaluated by running them against different benchmarks problems. However, the current-state-of-affairs is unsatisfactory.

As a first contribution, we demonstrated that any UAQ problem leveraging the extended DMER constraints introduced in (13) can be reduced to an “equivalent” UAQ problem containing only the traditional DMER constraints provided that a succinct representation of the current and past role activations is available. These results enable the use of existing UAQ solvers to support the enforcement of SoD requirements.

As a second contribution, by leveraging the asymptotic complexity analysis of the solving algorithms provided in (64), we proposed a methodology for evaluating existing benchmarks or designing new ones. The aforementioned algorithms play a key role in the proposed methodology, since they provide upper bounds on the asymptotic growth rate for UAQ solvers which, to the best of our knowledge, are the best upper bounds currently available in the literature. Yet, it must be noted that the proposed methodology will yield different,

improved results as soon as new complexity results will become available. To the best of our knowledge, this is the first approach to systematically assess the effectiveness of UAQ solvers. The methodology leads to benchmarks capable to (i) stress-test solvers along dimensions of the problem for which no polynomial-time technique is known but also (ii) to check their effectiveness, by determining whether they efficiently solve problems that are known to be solvable in polynomial time. We used our methodology to demonstrate that the benchmarks introduced in (64) are unsatisfactory.

We then introduced UAQ-Solve, a tool with double functionality: as a generator of benchmarks and as UAQ solver leveraging existing PMaxSAT solvers. By using UAQ-Solve, we applied our methodology to generate a novel suite of parametric benchmarks that allows for the systematic assessment of UAQ solvers over a number of relevant dimensions. These include problems for which no polynomial-time algorithm is known as well as problems for which polynomial-time algorithms do exist. The benchmarks proposed and used in this thesis could be improved in consequence to the improvement of the methodology due to new complexity results.

We used the new suite of benchmark problems as well as the benchmarks introduced in (64) to experimentally evaluate three solvers: 2D-Opt-Search (64) a search-based solver, 2D-Opt-CNF (64) that combines the reduction of the UAQ Decision Problem to SAT, a state-of-the-art SAT solver and a binary search, and UAQ-Solve. UAQ-Solve outperformed both 2D-Opt-Search and 2D-Opt-CNF in the vast majority of the benchmark considered.

Besides, we executed UAQ-Solve over the proposed benchmarks leveraging different PMaxSAT solvers to compare their performance. The result of this experiment is that, unsurprisingly, there is no single best solver, even for the same class of benchmark instances. In fact, the solver that best performs for small values of the parameter may be among the worst solvers for big values of the same parameter. Over different benchmarks, the solver that outperforms the others over a benchmark may be particularly inefficient over another benchmark. The choice of the PMaxSAT solver to use must be taken carefully according to the values of the UAQ problem dimensions. The use of UAQ-Solve over our benchmarks can suggest the best possible choice.

Finally, we made some investigations on incomplete solvers. We showed that incomplete solvers can perform similarly to complete ones over easy benchmarks. In addition, the experiments reveal that the quality of the solutions provided by an incomplete solver after 1 second over hard problems seems to be good; namely, overall, the incomplete solvers offer a good compromise between the optimality of the solutions and usability. Consequently, they are suitable for all the contexts in which usability can not be set aside in favor of security,

like healthcare, in our case. To the best of our knowledge, we are the first to propose the use of incomplete solvers over UAQ problems to tackle usability.

9.2 Future work

Future improvements to our work can be achieved from the point of view of both benchmarks generation and solving.

Regarding the benchmarks generation, the results of the evaluation of Pub for $o_p = max$ provided in Section 8.3.1 is not in line with our expectation. In fact, we did not expect exponential growth. It is then necessary to investigate this behavior.

A possible future work could consist in the support for constraints other than the MER constraints. The literature on constraints is huge, see for example (4; 5; 31; 49; 78; 87). Most of the constraints proposed are variants of SMER and DMER constraints. For example, in order to enforce SoD policies, in (4) Ahn and Sandhu suggest that permissions could be declared mutually exclusive instead of roles. In this way, a user cannot simultaneously be authorized for two or more mutually exclusive permissions. Instead, some works propose the use of languages to specify SoD constraints, such as RCL2000 (4; 5) and SoDA (52). It would be interesting to investigate whether and how these solutions can be leveraged in our work for constraint specification. Moreover, many RBAC variants have been proposed with different expressive features, such as temporal constraints on role activations (36), spatial constraints (57; 58), and spatio-temporal constraints (85). It is worth investigating whether and how our work could be expanded to include also these features.

From the point of view of problems solving, as a first future work, we can include other newer solvers in UAQ-Solve: obviously, the solvers that best performed at the MaxSAT Evaluation 2019 could be a good starting point. In addition, we mainly run experiments over instances encoded with permission-based optimization: we should deepen the analysis on joint-optimization.

As shown by the experimental results, the proper PMaxSAT solver to use must be carefully chosen. The approach of choosing a solver for a given class of instances is not satisfying: in fact, the results presented in Section 8.4 reveal that any solver over a benchmark can outperform the others for small values of the parameter under consideration and be among the worst performing solvers for big values of the same parameter (and vice-versa). It is then clear that the solver must be selected on a per-instance basis. As future work, we could include Machine Learning techniques to UAQ-Solve to automatically select the most

likely best solver among the ones available. From the experimental results, we can deduce that the following dimensions would be the most relevant features to consider:

- when $o_p = \min$:
 - $|P_{lb}|$: in our experiment, the best solver over *Plb_bigR* is Maxino for small values of $|P_{lb}|$ and MaxHS for big values of $|P_{lb}|$;
 - $|R|$: in our experiment, the best solvers over *R_bigPlb* are QMaxSAT and QMaxSATuc for small values of $|R|$ and MaxHS for medium values of $|R|$;
 - $\widehat{R_P}$: in our experiment, the best solver over *RPhat_bigPlb* is Maxino;
- when $o_p = \max$:
 - $|P_{ub}|$: in our experiment, the best solver over *Pub* is Loandra for small values of $|P_{ub}|$ and WPM1-2012 for big values of $|P_{ub}|$;
 - $|C|$: in our experiment, the best solvers over *C_bigR* are Maxino and QMaxSATuc for small values of $|C|$ and LMHS and MaxHS for big values of $|C|$;
 - \widehat{rs} : in our experiment, the best solver over *rshat_bigCt* is QMaxSATuc;
 - \widehat{t} : in our experiment, the best solver over *that_bigRPub* is Maxino.

However, we have to further improve our analysis, by generating and solving more UAQ problems, including more solvers and setting a higher timeout. The use of Machine Learning techniques may also be extended to the incomplete solvers, in order to select the solver that provides the best quality solution given a certain timeout.

Regarding the problem of solver selection, it is worth following the work of Matos et al. (61). Inspired by the success obtained by SATzilla¹ for SAT (91), the authors propose a first approach for a portfolio of algorithms for MaxSAT. This work leverages an oracle, which is able to predict the most suitable MaxSAT solver for a given instance. This solution can achieve significant performance improvements compared to existing solvers. Unfortunately, Matos et al. still have not extended their work to PMaxSAT and MaxSAT solvers, but in the paper they indicate this extension as future work.

Another important issue regarding the solvers performance is that, during the comparison of the different PMaxSAT solvers, we executed both the complete and the incomplete solvers with the default configurations. It is probable that they could have performed better using the proper settings. Finding the proper configurations to use is not easy. To this aim,

¹<http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/>

Ansótegui et al. improve the instance-specific algorithm configurator ISAC with the latest in portfolio technology (9). Experimental results on SAT reveal that this combination is a significant improvement in the ability to tune algorithms instance-specifically. The authors apply the methodology to a number of MaxSAT problem domains and show that the resulting solvers consistently outperform the best existing solvers. Given the good results obtained by Ansótegui et al., in the future we could also leverage the solution they propose to improve (hopefully) the solvers' performance.

We finally make a consideration on the encoding. In this dissertation, we treated the permissions as if they all have the same importance: in fact, in the encoding all permissions are given the same weight. However, it is reasonable that the resources have different values. For example, in the use case considered in this thesis, the information value depends on its confidentiality. The more sensitive the information is, the more impact is high in case of disclosure. Consequently, an interesting evolution of our work could be the extension of the well-known least privilege principle to the "least risk principle": in UAQ-Solve, this could be achieved by simply assigning to the permissions in $P \setminus P_{ub}$ a weight proportional to the risk related to the actions that can be performed through the permissions themselves.

Bibliography

- [1] A. M. A. Belov and J. Marques-Silva. Sat-based preprocessing for maxsat. volume 8312, page 96–111, 2013.
- [2] M. J. A. Belov and J. Marques-Silva. Formula preprocessing in mus extraction. volume 7795, page 108–123, 2013.
- [3] C. D. A. Morgado and J. Marques-Silva. Core-guided maxsat with soft cardinality constraints. *Annals of Mathematics and Artificial Intelligence*, 8656:564–573, 2014.
- [4] G. Ahn and R. S. Sandhu. The rsl99 language for role-based separation of duty constraints. In *4th Workshop on Role-Based Access Control*, page 43–54, 1999.
- [5] G. Ahn and R. S. Sandhu. Role-based authorization constraints specification. In *ACM Transactions on Information and System Security*, volume 3, page 207–226, 2000.
- [6] N. Alon, B. Awerbuch, and Y. Azar. The online set cover problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 100–105, 2003.
- [7] M. Alviano and C. Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. volume 16, page 533–551, 2016.
- [8] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy. Improving sat-based weighted maxsat solvers. In M. Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012.
- [9] C. Ansótegui, J. Gabas, Y. Malitsky, and M. Sellmann. Maxsat by improved instance-specific algorithm configuration. *Artificial Intelligence*, 235:26–39, 2016.
- [10] A. Armando and G. Gazzarata. RBAC user query authorization problem: Maxsat instances. *MaxSAT Evaluation*, 2018.
- [11] A. Armando, G. Gazzarata, and F. Turkmen. Aqua: An efficient solver for the user authorization query problem. Accepted at SACMAT 2020.
- [12] A. Armando, G. Gazzarata, and F. Turkmen. Benchmarking uaq solvers. Accepted at SACMAT 2020.
- [13] A. Armando, S. Ranise, F. Turkmen, and B. Crispo. Efficient run-time solving of RBAC user authorization queries: pushing the envelope. In *Second ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 241–248, 2012.

- [14] G. Audemard and L. Simon. Predicting learnt clauses quality in modern sat solvers. In *International Joint Conference on Artificial Intelligence*, page 399–404, 2009.
- [15] O. M. B. Andres, B. Kaufmann and T. Schaub. Unsatisfiability-based optimization in clasp. page 211–221, 2012.
- [16] J. M. D. P. P. B. Blobel, R. Nordberg. Modelling privilege management and access control. *Int. J. Med. Inform.*, 75(8):597–623, 2006.
- [17] O. Bailleux and Y. Boufkhad. Efficient cnf encoding of boolean cardinality constraints. volume 2833, page 108–122, 2003.
- [18] D. A. Basin, S. J. Burri, and G. Karjoh. Dynamic Enforcement of Abstract Separation of Duty Constraints. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS '09), Saint-Malo, France*, volume 5789 of *LNCS*, pages 250–267. Springer, September 2009.
- [19] J. Berg and M. Jarvisalo. Weight-aware core extraction in sat-based maxsat solving. volume 10416, pages 652–670, 2017.
- [20] D. L. Berre and A. Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.
- [21] A. Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.
- [22] B. Blobel. Security and privacy services in bling trustworthy personal health. *Stud. Health Technol. Inform.*, 179:203–217, 2012.
- [23] F. M. C. Ansótegui. Mapping problems with finite-domain variables to problems with boolean variables. page 1–15, 2004.
- [24] J. L. C. Ansótegui, M.L. Bonet. Solving (weighted) partial maxsat through satisfiability testing. volume 5584, 2009.
- [25] M. B. C. Ansótegui and J. Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196:77–105, 2013.
- [26] P. J. Chase. Algorithm 382: Combinations of m out of n objects [g6]. *Communications of the Association for Computing Machinery*, 13(6):368, 1970.
- [27] L. Chen and J. Crampton. Inter-domain role mapping and least privilege. In *SACMAT*, pages 157–162, 2007.
- [28] L. Chen and J. Crampton. Set covering problems in role-based access control. In *Proceedings of the 14th European conference on Research in computer security, ESORICS'09*, pages 689–704, 2009.
- [29] D. D. Clark and D. R. Wilson. A comparision of commercial and military computer security policies. In *IEEE Symposium on Security and Privacy*, page 184–194, 1987.
- [30] O. X. T. Committee, 2013. eXtensible Access Control Markup Language (XACML).

- [31] J. Crampton. Specifying and enforcing constraints in role-based access control. In *SACMAT*, pages 43–50. ACM, 2003.
- [32] J. Crampton and H. Khamhametu. A framework for enforcing constrained RBAC policies. In *CSE (3)*, pages 195–200, 2009.
- [33] J. Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. PhD thesis, University of Toronto, 2013.
- [34] J. Davies and F. Bacchus. Solving maxsat by solving a sequence of simpler sat instances. In *Lecture Notes in Computer Science*, volume 6129, page 225–239, 2011.
- [35] S. Du and J. B. D. Joshi. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In *SACMAT*, pages 228–236, 2006.
- [36] P. A. B. E. Bertino and E. Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4:191–233, August 2001.
- [37] N. Eén and N. Sorensson. An extensible sat-solver. In *Lecture Notes in Computer Science*, volume 2919, page 502–518, 2003.
- [38] P. S. F. Bacchus, M. Jarvisalo and A. Hyttinen. Reduced cost fixing in maxsat. In *Lecture Notes in Computer Science*, volume 10416, 2017.
- [39] D. F. Ferraiolo and D. R. Kuhn. Role-based access controls. In *15th National Computer Security Conference*, pages 554–563, 1992.
- [40] R. Ferrini and E. Bertino. Supporting RBAC with XACML+OWL. In *SACMAT*, pages 145–154, 2009.
- [41] S. Foley. The specification and implementation of ‘commercial’ security requirements including dynamic segregation of duties. In *ACM Conference on Computer and Communications Security (CCS-4)*, page 125–134, 1997.
- [42] Z. Fu. *Extending the power of boolean satisfiability: Techniques and applications*. PhD thesis, Princeton, 2007.
- [43] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.
- [44] S. Hayata and R. Hasegawa. Improvement in cnf encoding of cardinal constraints for weighted partial maxsat. *SIG-FPAI-B404. Japan Society for Artificial Intelligence*, page 80–84, 2015.
- [45] IBM. Cplex optimizer. <http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>, 2017.
- [46] A. N. S. Institute. American national standard for information technology - role based access control. <https://profsandhu.com/journals/tissec/ANSI+INCITS+359-2004.pdf>, 2004. ANSI INCITS 359-2004.

- [47] D. X. C. T. J. Lu, Z. Wang and J. Han. Towards an efficient approximate solution for the weighted user authorization query problem. *IEICE TRANSACTIONS on Information and Systems*, E100-D(8):1762–1769, August 2017.
- [48] A. G. J. Marques-Silva, J. Argelich and I. Lynce. Boolean lexicographic optimization: algorithms applications. *Annals of Mathematics and Artificial Intelligence*, 62(3-4):317–343, 2011.
- [49] T. Jaeger. On the increasing importance of constraints. In *ACM Workshop on Role-Based Access Control*, page 33–42, 1999.
- [50] R. M. Karp. Implicit hitting set problems and multi-genome alignment. In *Lecture Notes in Computer Science*, volume 6129, page 151, 2010.
- [51] N. Li, M. V. Tripunitara, and Z. Bizri. On mutually exclusive roles and separation-of-duty. *ACM Trans. Inf. Syst. Secur.*, 10, May 2007.
- [52] N. Li and Q. Wang. Beyond separation of duty: An algebra for specifying high-level security policies. *J. ACM*, 55(3), 2008.
- [53] J. Lu, J. B. D. Joshi, L. Jin, and Y. Liu. Towards complexity analysis of user authorization query problem in RBAC. *Computers & Security*, 48:116–130, 2015.
- [54] C. D. M. Alviano and F. Ricca. A maxsat algorithm using cardinality constraints of bounded size. page 2677–2683, 2015.
- [55] G. L. M. Davis and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.
- [56] H. F. M. Koshimura, T. Zhang and R. Hasegawa. Qmaxsat: A partial max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1-2):95–100, 2012.
- [57] B. C. M. L. Damiani, E. Bertino and P. Perlasca. GEO-RBAC: A spatially aware RBAC. *ACM Trans. Inf. Syst. Secur.*, 10, February 2007.
- [58] S. K. M. Nauman and X. Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, page 328–332, 2010.
- [59] X. Z. M. Xu, D. Wijesekera and D. Cooray. Towards session-aware RBAC administration and enforcement with XACML. In *IEEE International Symposium on Policies for Distributed Systems and Networks*, 2009.
- [60] J. Marques-Silva and J. Planes. On using unsatisfiability for solving maximum satisfiability. <https://arxiv.org/abs/0712.1097>, 2007.
- [61] P. J. Matos, J. Planes, F. Letombe, and J. Marques-Silva. A max-sat algorithm portfolio. In *ECAI*, pages 911–912, 2008.
- [62] B. K. R. I. Michael Gallaher, Alan O’Connor and G. T. (NIST). The economic impact of role-based access control. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=916549, 2002. ANSI INCITS 359-2004.

- [63] N. Mousavi. *Algorithmic Problems in Access Control*. PhD thesis, University of Waterloo, Canada, 2014.
- [64] N. Mousavi and M. V. Tripunitara. Mitigating the intractability of the user authorization query problem in role-based access control (RBAC). In *NSS*, pages 516–529, 2012.
- [65] D. E. Muller and F. P. Preparata. Bounds to complexities of networks for sorting and for switching. *Journal of the ACM (JACM)*, 22(2):195–201, 1975.
- [66] M. K. N. Uemura, H. Fujita and A. Zha. A sat encoding of pseudo-boolean constraints based on mixed radix. *SIG-FPAI-B506. Japan Society for Artificial Intelligence*, page 12–17, 2017.
- [67] N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided maxsat resolution. page 2717–2723, 2014.
- [68] M. J. Nash and K. R. Poland. Some conundrums concerning separation of duty. In *IEEE Symposium on Research in Security and Privacy*, page 201–209, 1990.
- [69] A. P. Punnen. In *The traveling salesman problem and its variations*. Springer, 2007.
- [70] V. M. R. Martins, S. Joshi and I. Lynce. Incremental cardinality constraints for maxsat. *Annals of Mathematics and Artificial Intelligence*, 8656:531–548, 2014.
- [71] H. L. F. Ravi S. Sandhu, Edward J. Coynek and C. E. Youmank. Role-based access control models. *Proceedings of the IEEE*, 29(2):38–47, 1996.
- [72] R. M. S. Joshi and V. M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In *Lecture Notes in Computer Science*, volume 9255, page 200–209, 2015.
- [73] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.
- [74] R. Sandhu. Transaction control expressions for separation of duties. In *Fourth Annual Computer Security Applications Conference (ACSAC'88)*, 1988.
- [75] R. Sandhu. Separation of duties in computerized information systems. In *IFIP WG11.3 Workshop on Database Security*, 1990.
- [76] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.
- [77] R. Sandhu and S. Jajodia. Integrity mechanisms in database management systems. In *NIST-NCSC National Computer Security Conference*, page 526–540, 1990.
- [78] T. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *10th Computer Security Foundations Workshop*, page 183–194, 1997.
- [79] C. Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming (CP)*, pages 827–831, 2005.

- [80] L. G. S.N. Foley and X. Qian. A security model of dynamic labeling providing a tiered approach to verification. In *IEEE Symposium on Research in Security and Privacy*, page 142–153, 1996.
- [81] P. Stanica. Good lower and upper bounds on binomial coefficients. *Journal of inequalities in pure and applied mathematics*, 2:30, 2001.
- [82] P. S. T. Korhonen, J. Berg and M. Jarvisalo. Maxpre: An extended maxsat preprocessor. In *Lecture Notes in Computer Science*, volume 10491, 2017.
- [83] R. H. M. K. T. Ogawa, Y. Liu and H. Fujita. Modulo based cnf encoding of cardinality constraints and its application to maxsat solvers. In *International Conference on Tools with Artificial Intelligence*, page 9–17, 2013.
- [84] S. R. W. Terry Mayfield, J. Eric Roskos and J. M. Boone. Integrity in automated infomation systems. <http://www.iwar.org.uk/comsec/resources/standards/rainbow/C-TR-79-91.htm>, 1991. C TECHNICAL REPORT 79-91 for NIST.
- [85] M. Toahchoodee and I. Ray. On the formalization and analysis of a spatio-temporal role-based access control model. *Journal of Computer Security*, 19(3):399–452, 2011.
- [86] F. Turkmen. *Exploring Dynamic Constraint Enforcement and Efficiency in Access Control*. PhD thesis, University of Trento, Italy, 2 2012.
- [87] S. I. G. V. D. Gligor and D. F. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *IEEE Symposium on Research in Securityand Privacy*, page 172–183, 1998.
- [88] D. F. F. Vincent C. Hu and D. R. K. (NIST). Assessment of access control systems. <https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7316.pdf>, 2006. NIST Interagency Report 7316.
- [89] J. P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Inf. Process. Lett.*, 68(2):63–69, 1998.
- [90] G. T. Wickramaarachchi, W. H. Qardaji, and N. Li. An efficient framework for user authorization queries in RBAC systems. In *SACMAT*, pages 23–32, 2009.
- [91] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.
- [92] S. M. Z. Fu. On solving the partial max-sat problem. page 252–265, 2006.
- [93] Y. Zhang and J. B. D. Joshi. UAQ: a framework for user authorization query processing in RBAC extended with hybrid hierarchy and constraints. In *SACMAT*, pages 83–92, 2008.

Appendix A

Proof of Theorem 4.4.1

Theorem 4.4.1 *Let RP be an RBAC policy, $\sigma = (S, \alpha, \pi)$ be a valid state for RP and q is a query for session s . It can be shown that ρ is a solution to q in RP and $\sigma = (S, \alpha, \pi)$ iff ρ is a solution to q in RP' , where RP' is obtained from RP by replacing the sets of constraints C with $C[s, \sigma]$, where $C[s, \sigma]$ is a set of constraints in which MS-DMER, SS-HMER and MS-HMER for the particular session s are replaced with SS-DMER constraints whose rs and t varies with the state σ .*

Theorem 4.4.1 is demonstrated in Proof A.0.2. The latter leverages Lemma A.0.1, which is defined and proved below.

Lemma A.0.1 *For all sets A , B and C*

$$(A \setminus B) \cap C = (A \cap (B \cup C)) \setminus (A \cap B) \quad (\text{A.1})$$

Proof A.0.1 *We know that $X \setminus Y = X \setminus (X \cap Y)$ holds for all sets X and Y . From this it follows that*

$$(A \cap (B \cup C)) \setminus B = (A \cap (B \cup C)) \setminus ((A \cap (B \cup C)) \cap B) \quad (\text{A.2})$$

The left hand side of (A.2) can be rewritten as follows:

$$(A \cap (B \cup C)) \setminus B = (A \setminus B) \cap ((B \cup C) \setminus B) = \quad (\text{A.3})$$

$$= (A \setminus B) \cap (C \setminus B) = \quad (\text{A.4})$$

$$= (A \setminus B) \cap C \quad (\text{A.5})$$

By observing that

$$((A \cap (B \cup C)) \cap B) = (A \cap ((B \cup C) \cap B)) = \quad (\text{A.6})$$

$$= (A \cap B) \quad (\text{A.7})$$

the right hand side of (A.2) can be rewritten to the right hand side of (A.1).

Proof A.0.2 (Theorem 4.4.1) We focus the case in which RP contains a single constraint of the form $\text{MS-DMER}(rs, t)$ only. The other cases can be dealt with analogously. Let $\sigma = (S, \alpha, \pi)$, q a query for session s , $\rho \subseteq R_{\text{user}(s)}$ a solution to q in RP. We show that ρ is also a solution to q in RP' , where RP' is obtained from RP by replacing the constraint $\text{MS-DMER}(rs, t)$ with $\text{SS-DMER}(rs \setminus ur, t - |rs \cap ur|)$, where $ur = \bigcup_{s' \in S_{\text{user}(s)} \setminus \{s\}} \alpha(s')$. This amounts to showing that ρ is such that for all $s'' \in S$

$$|rs \setminus ur \cap \alpha[s \leftarrow \rho](s'')| < t - |rs \cap ur| \quad (\text{A.8})$$

where $ur = \bigcup_{s' \in S_{\text{user}(s)} \setminus \{s\}} \alpha[s \leftarrow \rho](s')$.

If $s'' = s$,¹ then by using (A.1), (A.8) can be rewritten to

$$|rs \cap (ur \cup \rho) \setminus (rs \cap ur)| < t - |rs \cap ur|$$

which in turn can be rewritten to

$$|(rs \cap \rho) \setminus (rs \cap ur)| < t - |rs \cap ur| \quad (\text{A.9})$$

Now, we know that $(S, \alpha[s \leftarrow \rho], \pi[s \leftarrow \rho])$ satisfies $\text{MS-DMER}(rs, t)$. Therefore, for all $u \in U$

$$|rs \cap \bigcup_{s' \in S_u} \alpha[s \leftarrow \rho](s')| < t$$

From this it follows

$$|rs \cap (\rho \cup \bigcup_{s' \in S_{\text{user}(s)} \setminus \{s\}} \alpha(s'))| < t$$

which can be rewritten as

$$|(rs \cap \rho) \cup (rs \cap ur)| < t \quad (\text{A.10})$$

¹The case $s'' \neq s$ is analogous.

Since $X \cup Y = (X \setminus Y) \cup Y$, then (A.10) is equivalent to

$$|((rs \cap \rho) \setminus (rs \cap ur)) \cup (rs \cap ur)| < t$$

which leads to (A.9) by observing that $|X \cup Y| = |X| + |Y|$ if $X \cap Y = \emptyset$.

Appendix B

WCNF encodings

This appendix presents the different WCNF encoding of the instance shown in Listing 7.2 resulting from different encoding configuration. In particular, in Listing B.1

- $o_p = \text{min}$;
- $o_r = \text{any}$;
- $\text{pri} = p$,

while in Listing B.2

- $o_p = \text{max}$;
- $o_r = \text{any}$;
- $\text{pri} = p$.

In both the cases, the optimization is then permission-based. Listing B.1 was already shown and widely discussed in Section 7.1 (Listing 7.3), but is reported here for completeness.

Listing B.1 WCNF encoding for $o_p = \text{min}$, $o_r = \text{any}$, and $\text{pri} = p$

```
1 p wcnf 15 41 41
2 41 -1 15 0
3 41 -2 10 0
4 41 -2 12 0
5 41 -2 14 0
6 41 -3 6 0
7 41 -3 7 0
```

```

8 41 -3 8 0
9 41 -3 9 0
10 41 -3 11 0
11 41 -3 13 0
12 41 -3 14 0
13 41 -3 15 0
14 41 -4 7 0
15 41 -4 8 0
16 41 -4 9 0
17 41 -4 11 0
18 41 -4 12 0
19 41 -4 13 0
20 41 -5 6 0
21 41 -5 10 0
22 41 -6 3 5 0
23 41 -7 3 4 0
24 41 -8 3 4 0
25 41 -9 3 4 0
26 41 -10 2 5 0
27 41 -11 3 4 0
28 41 -12 2 4 0
29 41 -13 3 4 0
30 41 -14 2 3 0
31 41 -15 1 3 0
32 41 -2 -4 0
33 41 6 0
34 41 7 0
35 41 -8 0
36 1 -9 0
37 1 -10 0
38 1 -11 0
39 1 -12 0
40 1 -13 0
41 1 -14 0
42 1 -15 0

```

Listing B.2 WCNF encoding for $o_p = \max$, $o_r = \text{any}$, and $pri = p$

```

1 p wcnf 15 41 41
2 41 -1 15 0
3 41 -2 10 0
4 41 -2 12 0

```

```

5 41 -2 14 0
6 41 -3 6 0
7 41 -3 7 0
8 41 -3 8 0
9 41 -3 9 0
10 41 -3 11 0
11 41 -3 13 0
12 41 -3 14 0
13 41 -3 15 0
14 41 -4 7 0
15 41 -4 8 0
16 41 -4 9 0
17 41 -4 11 0
18 41 -4 12 0
19 41 -4 13 0
20 41 -5 6 0
21 41 -5 10 0
22 41 -6 3 5 0
23 41 -7 3 4 0
24 41 -8 3 4 0
25 41 -9 3 4 0
26 41 -10 2 5 0
27 41 -11 3 4 0
28 41 -12 2 4 0
29 41 -13 3 4 0
30 41 -14 2 3 0
31 41 -15 1 3 0
32 41 -2 -4 0
33 41 6 0
34 41 7 0
35 41 -8 0
36 1 9 0
37 1 10 0
38 1 11 0
39 1 12 0
40 1 13 0
41 1 14 0
42 1 15 0

```

Rows 1 to 35 are the same in Listings B.1 and B.2. In fact, they differ only for the soft clauses (rows 36 - 42):

- in Listing B.1, since $o_p = \min$ the soft clauses require that the variables 9, 10, 11, 12, 13, 14, and 15 (representing the activation of extra permissions) are false (see the '-' in front of the variables);
- in Listing B.2, since $o_p = \max$ the soft clauses require that the variables 9, 10, 11, 12, 13, 14, and 15 are true (there is not a '-' in front of the variables);

Listings B.1 and B.2 are related to permission-based optimization. Instead, Listings B.3, B.4, B.5, and B.6 are related to joint optimization. In particular, the priority is permission optimization. The WCNF encodings contain

$$nbclauses = 46$$

clauses, namely 5 more than Listing B.1: this is due to the $|R_u| = 5$ clauses added for the conditions on extra roles. The clauses on the activation of extra permissions have weight

$$W_p = |R_u| + 1 = 5 + 1 = 6,$$

while the clauses on the activation of extra roles have weight

$$W_r = 1.$$

The hard clauses have weight

$$W_h = nbclauses + ((|R_u| + 1) \times (|P_{ub}| - |P_{lb}|)) = 46 + ((5 + 1) \times (9 - 2)) = 88.$$

We use the aforementioned formula to ensure that W_h is greater than the sum of the weights given to the soft clauses. However, any other formula granting this condition would have been adequate.

Listing B.3 presents the WCNF encoding resulting from the following configuration:

- $o_p = \min$;
- $o_r = \min$;
- $pri = p$.

With respect to Listings B.1, we notice the different weight $W_h = 88$ for the hard clauses (rows 2 - 35) and the different weight $W_p = 6$ for the clauses related to the activation of extra permissions (rows 36 - 42). Then there are additional clauses, namely the ones related to

the activation of extra roles (rows 43 - 47), with weight $W_r = 1$. Since $o_r = \text{min}$, the clauses indicates that we prefer that the variables representing the roles activation (1, 2, 3, 4, 5) are false. In fact, there is a '-' in front of them.

Listing B.3 WCNF encoding for $o_p = \text{min}$, $o_r = \text{min}$, and $pri = p$

```

1 p wcnf 15 46 88
2 88 -1 15 0
3 88 -2 10 0
4 88 -2 12 0
5 88 -2 14 0
6 88 -3 6 0
7 88 -3 7 0
8 88 -3 8 0
9 88 -3 9 0
10 88 -3 11 0
11 88 -3 13 0
12 88 -3 14 0
13 88 -3 15 0
14 88 -4 7 0
15 88 -4 8 0
16 88 -4 9 0
17 88 -4 11 0
18 88 -4 12 0
19 88 -4 13 0
20 88 -5 6 0
21 88 -5 10 0
22 88 -6 3 5 0
23 88 -7 3 4 0
24 88 -8 3 4 0
25 88 -9 3 4 0
26 88 -10 2 5 0
27 88 -11 3 4 0
28 88 -12 2 4 0
29 88 -13 3 4 0
30 88 -14 2 3 0
31 88 -15 1 3 0
32 88 -2 -4 0
33 88 6 0
34 88 7 0
35 88 -8 0
36 6 -9 0

```

```

37 6 -10 0
38 6 -11 0
39 6 -12 0
40 6 -13 0
41 6 -14 0
42 6 -15 0
43 1 -1 0
44 1 -2 0
45 1 -3 0
46 1 -4 0
47 1 -5 0

```

Listing B.4 presents the WCNF encoding resulting from the following configuration:

- $o_p = \min$;
- $o_r = \max$;
- $pri = p$.

The only difference with Listing B.3 is that in Listing B.4 the clauses in rows 43 - 47 do not present '-' in front of the variables. In fact, in this case $o_r = \max$, thus we want to maximize the activation of roles.

Listing B.4 WCNF encoding for $o_p = \min$, $o_r = \max$, and $pri = p$

```

1 p wcnf 15 46 88
2 88 -1 15 0
3 88 -2 10 0
4 88 -2 12 0
5 88 -2 14 0
6 88 -3 6 0
7 88 -3 7 0
8 88 -3 8 0
9 88 -3 9 0
10 88 -3 11 0
11 88 -3 13 0
12 88 -3 14 0
13 88 -3 15 0
14 88 -4 7 0
15 88 -4 8 0
16 88 -4 9 0
17 88 -4 11 0

```

```

18 88 -4 12 0
19 88 -4 13 0
20 88 -5 6 0
21 88 -5 10 0
22 88 -6 3 5 0
23 88 -7 3 4 0
24 88 -8 3 4 0
25 88 -9 3 4 0
26 88 -10 2 5 0
27 88 -11 3 4 0
28 88 -12 2 4 0
29 88 -13 3 4 0
30 88 -14 2 3 0
31 88 -15 1 3 0
32 88 -2 -4 0
33 88 6 0
34 88 7 0
35 88 -8 0
36 6 -9 0
37 6 -10 0
38 6 -11 0
39 6 -12 0
40 6 -13 0
41 6 -14 0
42 6 -15 0
43 1 1 0
44 1 2 0
45 1 3 0
46 1 4 0
47 1 5 0

```

Listing B.5 presents the WCNF encoding resulting from the following configuration:

- $o_p = \text{max}$;
- $o_r = \text{min}$;
- $\text{pri} = p$.

Listing B.5 differs from Listing B.3 only for the absence of '-' in front of the variables in rows 36 - 42. In fact, in this case $o_p = \text{max}$, thus we wish to maximize the activation of permissions.

Listing B.5 WCNF encoding for $o_p = \max$, $o_r = \min$, and $pri = p$

```

1 p wcnf 15 46 88
2 88 -1 15 0
3 88 -2 10 0
4 88 -2 12 0
5 88 -2 14 0
6 88 -3 6 0
7 88 -3 7 0
8 88 -3 8 0
9 88 -3 9 0
10 88 -3 11 0
11 88 -3 13 0
12 88 -3 14 0
13 88 -3 15 0
14 88 -4 7 0
15 88 -4 8 0
16 88 -4 9 0
17 88 -4 11 0
18 88 -4 12 0
19 88 -4 13 0
20 88 -5 6 0
21 88 -5 10 0
22 88 -6 3 5 0
23 88 -7 3 4 0
24 88 -8 3 4 0
25 88 -9 3 4 0
26 88 -10 2 5 0
27 88 -11 3 4 0
28 88 -12 2 4 0
29 88 -13 3 4 0
30 88 -14 2 3 0
31 88 -15 1 3 0
32 88 -2 -4 0
33 88 6 0
34 88 7 0
35 88 -8 0
36 6 9 0
37 6 10 0
38 6 11 0
39 6 12 0
40 6 13 0
41 6 14 0

```

```

42 6 15 0
43 1 -1 0
44 1 -2 0
45 1 -3 0
46 1 -4 0
47 1 -5 0

```

Listing B.6 presents the WCNF encoding resulting from the following configuration:

- $o_p = \text{max}$;
- $o_r = \text{max}$;
- $\text{pri} = p$.

Similarly to the previous cases, Listing B.5 and Listing B.6 are different only in the clauses in rows 43 - 47. In case of Listing B.6, there is not a '-' in front of the variables. In fact, $o_r = \text{max}$, meaning that we wish to maximize the activation of roles.

Listing B.6 WCNF encoding for $o_p = \text{max}$, $o_r = \text{max}$, and $\text{pri} = p$

```

1 p wcnf 15 46 88
2 88 -1 15 0
3 88 -2 10 0
4 88 -2 12 0
5 88 -2 14 0
6 88 -3 6 0
7 88 -3 7 0
8 88 -3 8 0
9 88 -3 9 0
10 88 -3 11 0
11 88 -3 13 0
12 88 -3 14 0
13 88 -3 15 0
14 88 -4 7 0
15 88 -4 8 0
16 88 -4 9 0
17 88 -4 11 0
18 88 -4 12 0
19 88 -4 13 0
20 88 -5 6 0
21 88 -5 10 0
22 88 -6 3 5 0

```

```

23 88 -7 3 4 0
24 88 -8 3 4 0
25 88 -9 3 4 0
26 88 -10 2 5 0
27 88 -11 3 4 0
28 88 -12 2 4 0
29 88 -13 3 4 0
30 88 -14 2 3 0
31 88 -15 1 3 0
32 88 -2 -4 0
33 88 6 0
34 88 7 0
35 88 -8 0
36 6 9 0
37 6 10 0
38 6 11 0
39 6 12 0
40 6 13 0
41 6 14 0
42 6 15 0
43 1 1 0
44 1 2 0
45 1 3 0
46 1 4 0
47 1 5 0

```

In case $pri = r$, the WCNF encodings would contain

$$nbclauses = 46$$

clauses. The clauses on the activation of extra permissions would have weight

$$W_p = 1,$$

while the clauses on the activation of extra roles would have weight

$$W_r = |P_{ub} \setminus P_{lb}| + 1 = 9 - 2 + 1 = 8$$

The hard clauses would have weight

$$W_h = nbclauses + (|R_u| \times (|P_{ub}| - |P_{lb}| + 1)) = 46 + (5 \times (9 - 2 + 1)) = 86$$

We use the aforementioned formula to ensure that W_h is greater than the sum of the weights given to the soft clauses. However, any other formula granting this condition would have been adequate.

Even if not considered in this thesis, for the sake of completeness, we report an example of encoding for joint optimization with the activation of roles as priority. In particular, Listing B.7 presents the WCNF encoding resulting from the following configuration:

- $o_p = \text{min}$;
- $o_r = \text{min}$;
- $\text{pri} = r$.

Unlike the previous cases, the clauses with the lowest weight are the ones related to the activation of extra permissions (rows 36 - 42, with $W_p = 1$). Instead, the clauses related to the activation of extra roles (rows 43 - 47) have weight $W_r = 8$. In this case $o_p = o_r = \text{min}$, in fact all the variables of the clauses in rows 36 - 47 have a '-' in front of them, meaning that we wish to minimize the activation of both roles and permissions.

Listing B.7 WCNF encoding for $o_p = \text{min}$, $o_r = \text{min}$, and $\text{pri} = r$

```

1 p wcnf 15 46 86
2 86 -1 15 0
3 86 -2 10 0
4 86 -2 12 0
5 86 -2 14 0
6 86 -3 6 0
7 86 -3 7 0
8 86 -3 8 0
9 86 -3 9 0
10 86 -3 11 0
11 86 -3 13 0
12 86 -3 14 0
13 86 -3 15 0
14 86 -4 7 0
15 86 -4 8 0
16 86 -4 9 0
17 86 -4 11 0
18 86 -4 12 0
19 86 -4 13 0
20 86 -5 6 0

```

```

21 86 -5 10 0
22 86 -6 3 5 0
23 86 -7 3 4 0
24 86 -8 3 4 0
25 86 -9 3 4 0
26 86 -10 2 5 0
27 86 -11 3 4 0
28 86 -12 2 4 0
29 86 -13 3 4 0
30 86 -14 2 3 0
31 86 -15 1 3 0
32 86 -2 -4 0
33 86 6 0
34 86 7 0
35 86 -8 0
36 1 -9 0
37 1 -10 0
38 1 -11 0
39 1 -12 0
40 1 -13 0
41 1 -14 0
42 1 -15 0
43 8 -1 0
44 8 -2 0
45 8 -3 0
46 8 -4 0
47 8 -5 0

```

The WCNF encodings resulting from the remaining configurations, namely

- $o_p = \min$, $o_r = \max$, and $pri = r$;
- $o_p = \max$, $o_r = \min$, and $pri = r$;
- $o_p = \max$, $o_r = \max$, and $pri = r$

can be obtained from Listing B.7 by removing the '-' in front of the variables in rows 36 - 42 and 43 - 47 when needed, according to o_r and o_p .

Appendix C

Encoding time and number of variables and clauses

For the sake of completeness, this annex presents plots representing

- The median encoding time;
- The number of variables; and
- The number of clauses

of the benchmarks designed in Chapter 6. In particular, Section C.1 presents the plots for the benchmarks with $o_p = \min$, while Section C.2 presents the plot for the benchmarks with $o_p = \max$.

All the plots presented below show that the median encoding time is acceptable. However, in principle, the problems could be in part encoded once. At run-time it would be enough to add the clauses related to DMER constraints¹ and to the query.

In earlier research, we thought that the problem complexity could depend only on the number of variables and clauses. However, we have evidence that this does not hold, as we will show below.

¹If MS-DMER, SS-HMER and MS-HMER constraints are used they must be properly reduced to SS-DMER constraints, see Section 4.4

C.1 Benchmarks with $o_p = \min$

Benchmarks parametric in P_{lb}

Figure C.1 shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in $|P_{lb}|$, namely Plb_bigR and Plb_smallR . In both the cases, the number of variables and clauses are constant. However, as already shown in Figure 8.4a, UAQ-Solve over Plb_bigR exhibits an exponential behavior. This is a first clear empirical proof that complexity is not related only to the number of clauses and variables.

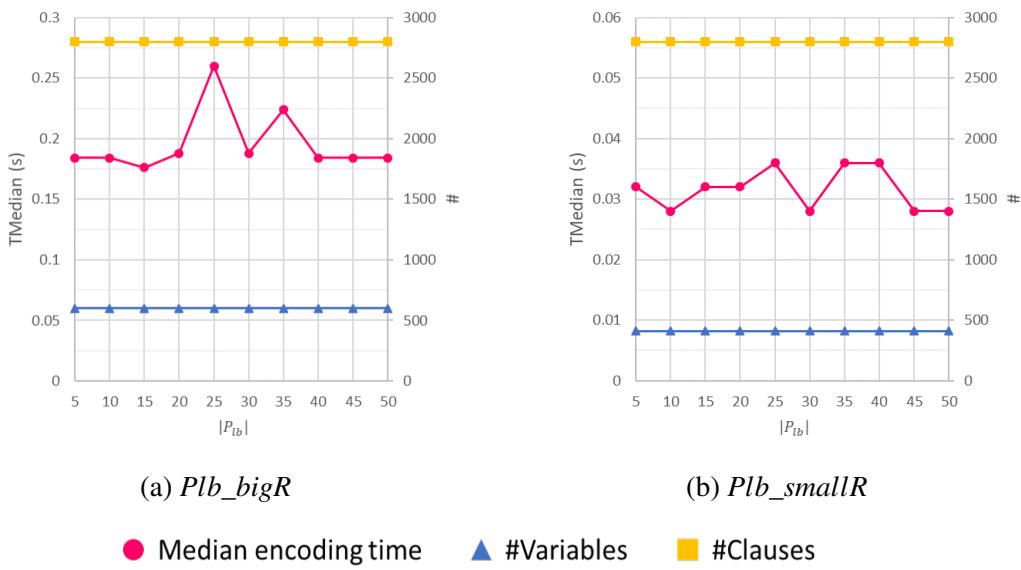


Figure C.1 Median encoding time, number of variables and number of clauses in the benchmarks parametric in P_{lb} with $o_p = \min$

Benchmarks parametric in $|R|$

Figure 8.5a shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in $|R|$, namely R_bigPlb and $R_smallPlb$. The curves representing the number of variables are identical; the same holds for the number of clauses. Nevertheless, UAQ-Solve solves R_bigPlb in exponential time, while over $R_smallPlb$ the median solving even decreases with the increase in $|R|$.

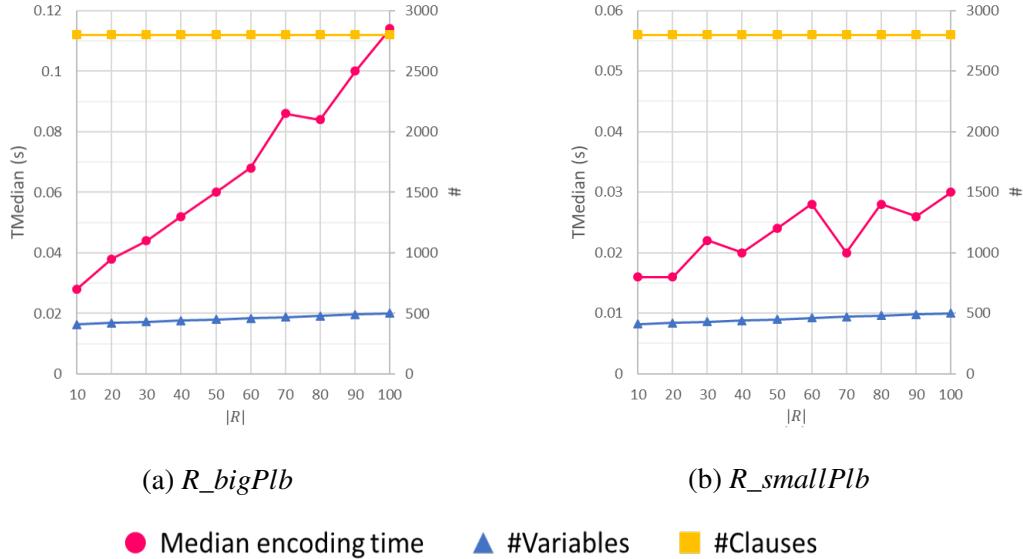


Figure C.2 Median encoding time, number of variables and number of clauses in the benchmarks parametric in $|R|$ with $o_p = \min$

Benchmarks parametric in \widehat{R}_P

Figure C.3c shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in \widehat{R}_P , namely $RPhat_bigPlb$, $RPhat_medPlb$, and $RPhat_smallPlb$. The number of variables is constant and identical for the three benchmarks. Also the number of clauses is the same for the benchmarks. However, as shown in Figure 8.6a the median solving time over the benchmarks is a polynomial whose degree is $|P_{lb}|$.

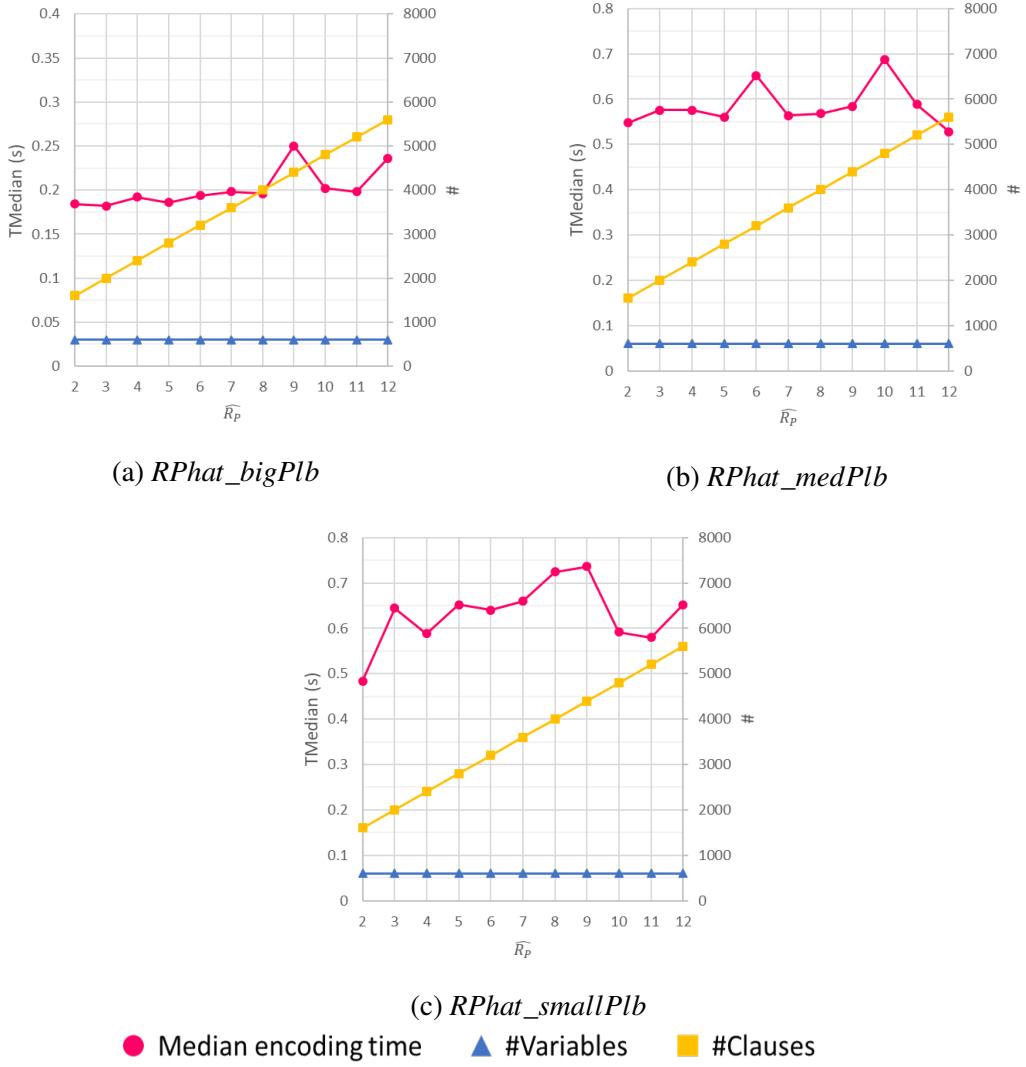


Figure C.3 Median encoding time, number of variables and number of clauses in the benchmarks parametric in \widehat{R}_P with $o_p = \min$

Benchmarks parametric in $|P_{ub}|$, $|C|$, $|\widehat{rs}|$, and \widehat{t}

Figure C.4 shows the median encoding time, the number of variables and the number of clauses for the remaining benchmarks for $o_p = \min$, namely *Pub*, *C*, *rshat*, and *that*. For all these benchmarks the number of variables and clauses is higher than the ones shown in Figure C.2a. However UAQ-Solve solves the instances contained in *R_bigPlb* in exponential time, while it quickly solves all the instances contained in these benchmarks. Figure C.4c is of particular interest: for $\widehat{rs} = 50$, the number of clauses is 198800 (compared to the 2800 clauses for *R_bigPlb*). Nevertheless, these instances are solved in less than 1 second.

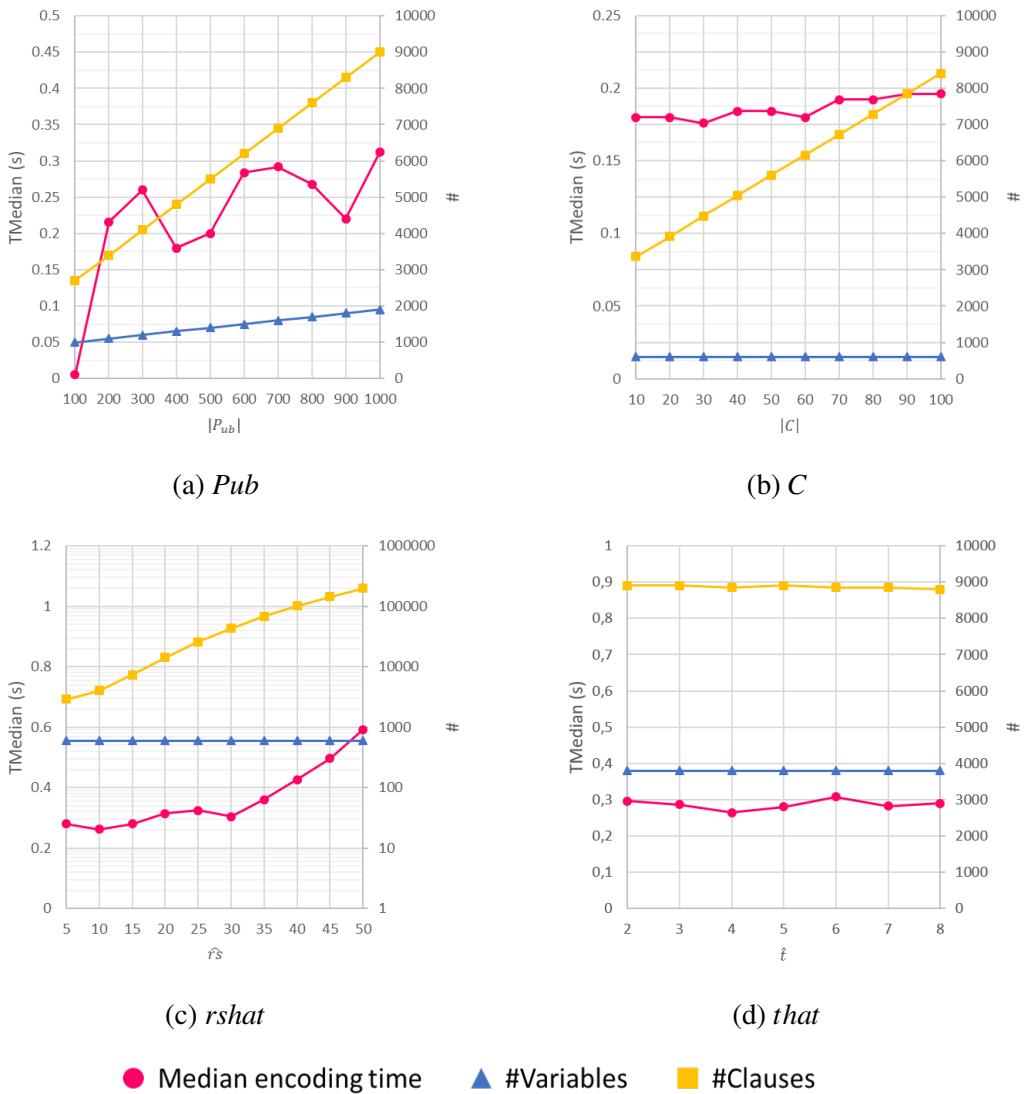


Figure C.4 Median encoding time, number of variables and number of clauses in the benchmarks parametric in $|P_{ub}|$, $|C|$, $|\hat{rs}|$, and \hat{t} with $o_p = \min$

C.2 Benchmarks with $o_p = \max$

Benchmarks parametric in $|R|$

Figure C.5 shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in $|R|$, namely R_bigCt and $R_smallCt$. The number of variables in these benchmarks is similar, while the number of clauses for $R_smallCt$ is more than 2000 greater than the number of clauses for $R_smallCt$. However UAQ-Solve

solves R_bigCt in exponential time and $R_smallCt$ in at most 10 milliseconds, as shown in Figure 8.16a.

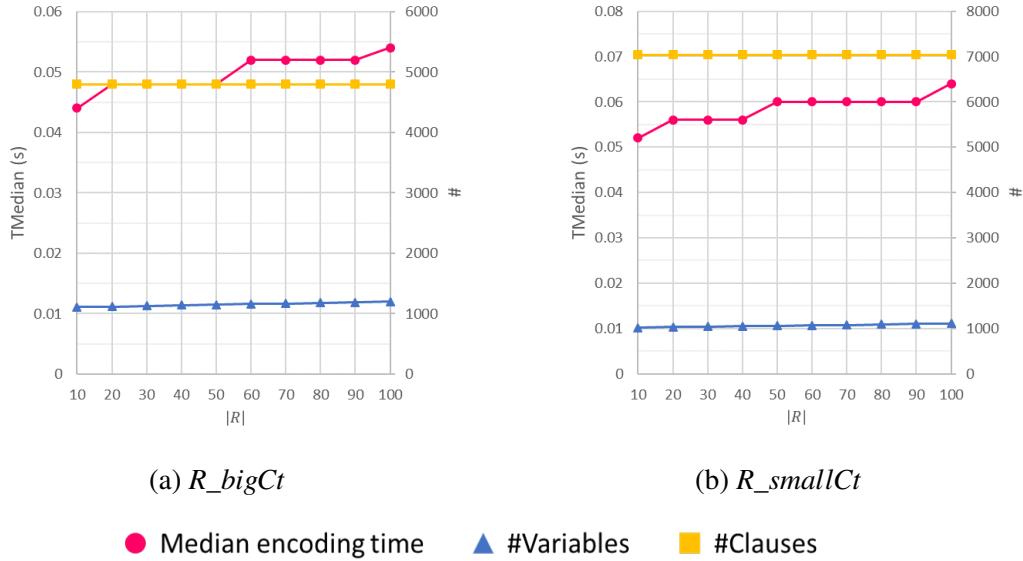


Figure C.5 Median encoding time, number of variables and number of clauses in the benchmarks parametric in $|R|$ with $o_p = \max$

Benchmarks parametric in $|C|$

Figure C.6 shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in $|C|$, namely C_bigR and C_smallR . The number of variables and clauses contained in the WCNF encoding of the instances in C_bigR are identical to the ones for C_smallR . Nevertheless, Figure 8.18a shows that UAQ-Solve solves C_bigR in exponential time and C_smallR in almost constant time.

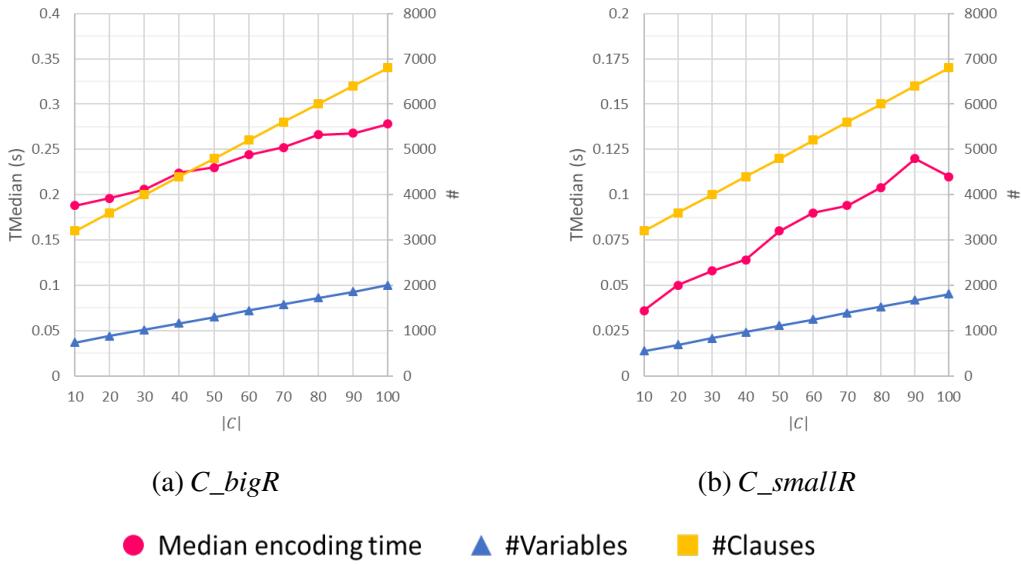


Figure C.6 Median encoding time, number of variables and number of clauses in the benchmarks parametric in $|C|$ with $o_p = \max$

Benchmarks parametric in \hat{t}

Figure C.7 shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in \hat{t} , namely *that_bigR* and *that_smallR*. Although the number of variables is constant and the number of clauses is always in the range [8800, 8900], UAQ-Solve exhibits an exponential growth over *that_bigR*, as shown in Figure 8.19a.

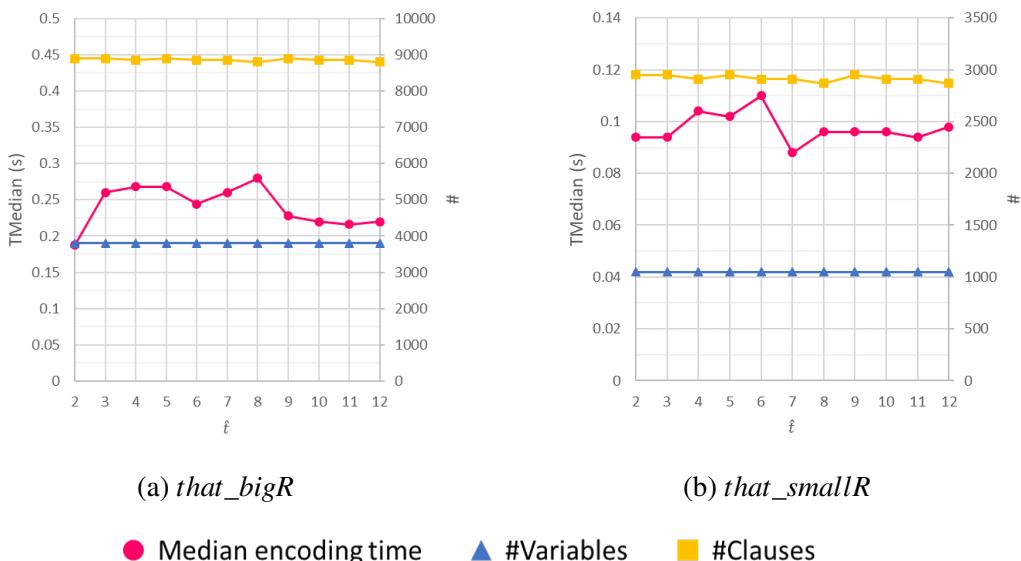


Figure C.7 Median encoding time, number of variables and number of clauses in the benchmarks parametric in \hat{t} with $o_p = \max$

Benchmarks parametric in \hat{rs}

Figure C.8 shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in \hat{rs} , namely *rshat_bigCt*, *rshat_medCt*, and *rshat_smallCt*.

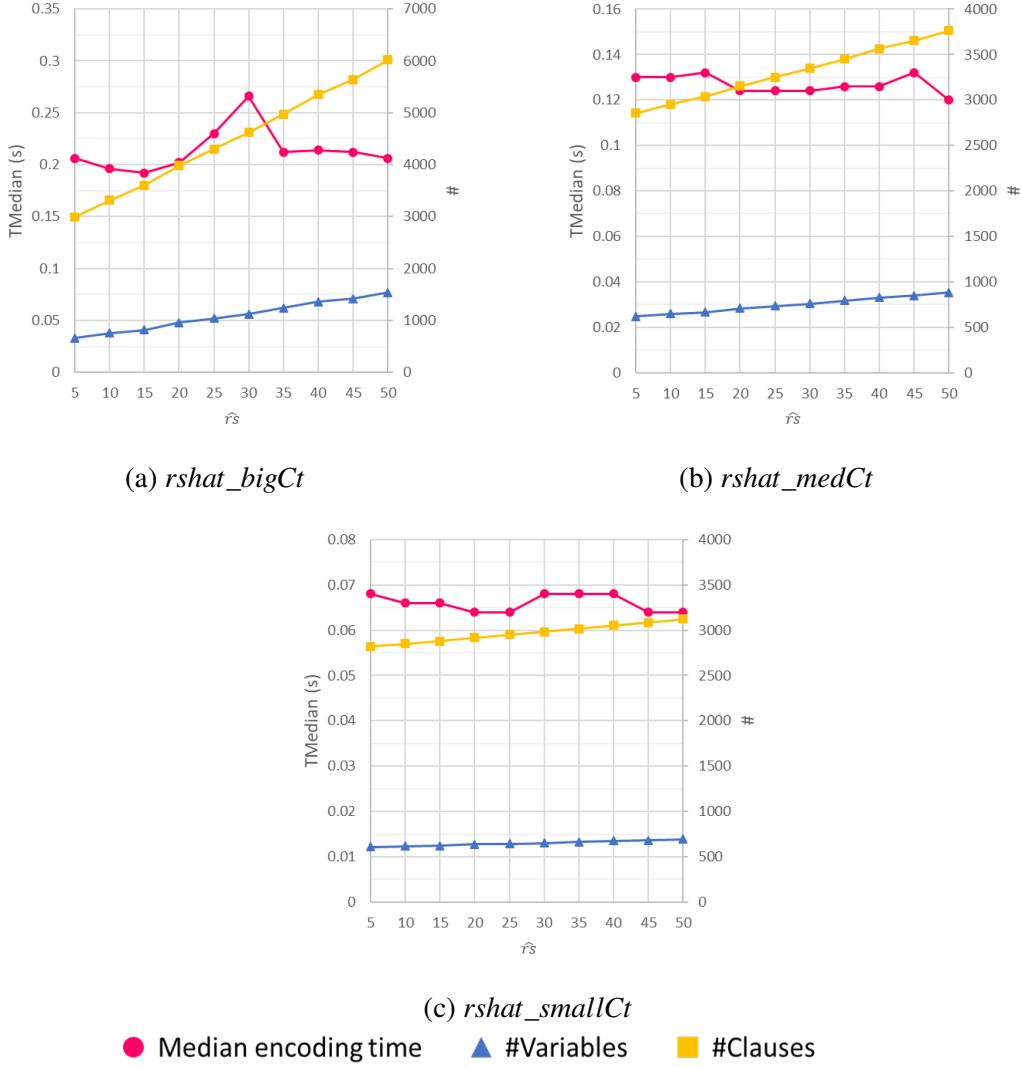


Figure C.8 Median encoding time, number of variables and number of clauses in the benchmarks parametric in \hat{rs} with $o_p = \max$

Benchmarks parametric in $|P_{ub}|$, $\widehat{R_P}$, and $|P_{lb}|$

Figure C.9 shows the median encoding time, the number of variables and the number of clauses for the benchmarks parametric in the remaining benchmarks for $o_p = \max$, namely

$|P_{ub}|$, $RPhat$, and Plb . Here we note that the number of clauses for $RPhat$ grows from 10800 to 82800, however all the instances in $RPhat$ are solved in less than 100 milliseconds.

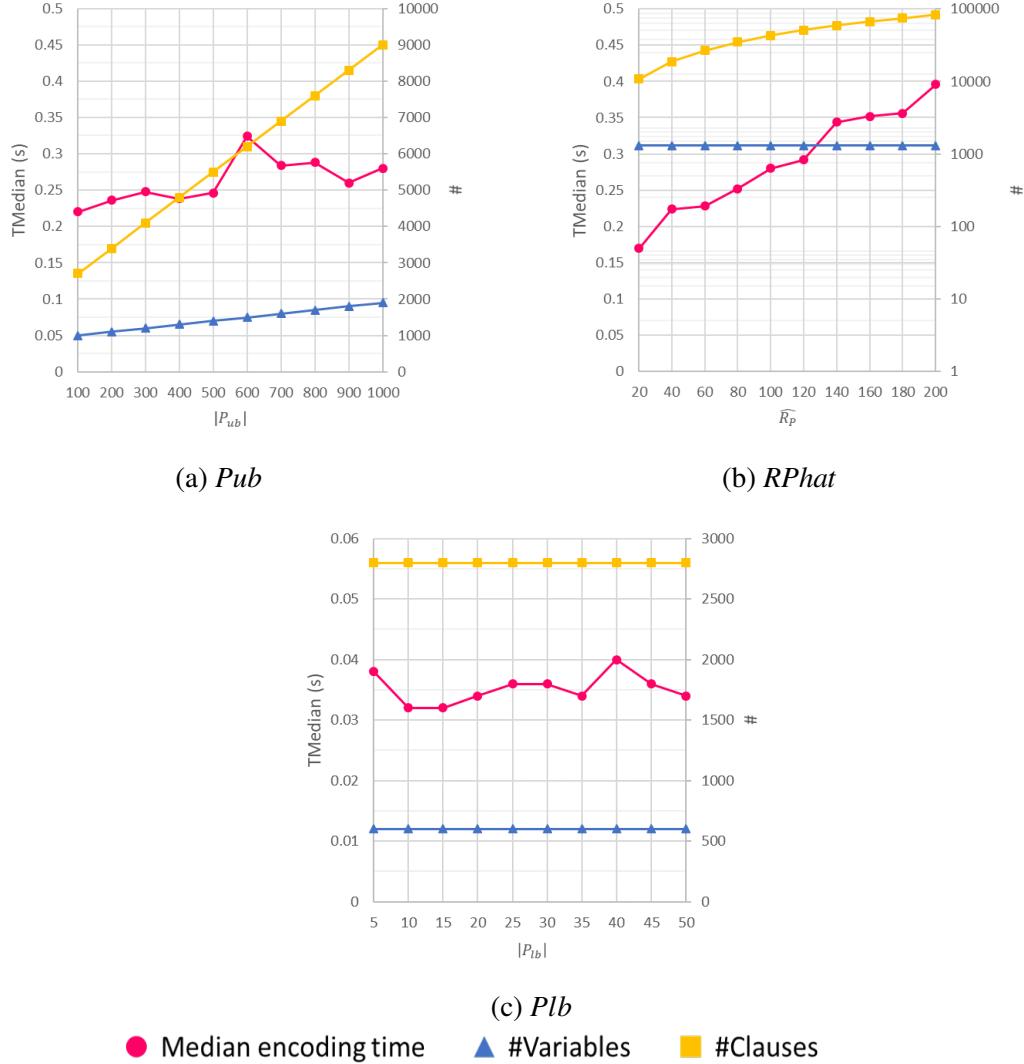


Figure C.9 Median encoding time, number of variables and number of clauses in the benchmarks parametric in $|P_{lb}|$ with $o_p = \max$